# EXHIBIT 1

# THE PHYSICAL DESIGN OF THE ILLIAC 6 SUPERCOMPUTING PLATFORM

BY

SEAN J. KELLER

B.S., Cornell University, 2002
M.Eng., Cornell University, 2004

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

# Abstract

An emerging class of problems that require realtime computation on high-bandwidth live data streams has created an opportunity for new high-performance computing architectures. By leveraging the availability of high-perfmance, low-power, commodity DSPs and FPGAs, we are able to design a novel reconfigurable supercomputing platform that satisfies the computational and bandwidth demands of the applications required to solve these modern problems. In this thesis, we present the architecture and physical design of this system, the ILLIAC 6.

# Acknowledgments

I would like to thank my advisor, Prof. Luddy Harrison, for his help and instruction during this project. His proficiency as a researcher and as a leader has made his goal of architecting, designing, and building a novel supercomputer a reality.

I would also like to thank the many students I have worked with in my research group. Without their efforts, much of my work would have been for naught.

I would also like to thank Meg for her assistance with editing this thesis and in for her loving support throughout the challenge of graduate school.

Finally, I thank both Analog Devices and Mentor Graphics for their assistance with this project. In particular I would like to thank the numerous engineers in the Mentor Graphics HyperLynx group who helped me with high-speed simulations.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| 8B/10B | 8-bit / 10-bit. |
| AC | Alternating Current. |
| ASCII | American Standard Code for Information Interchange. |
| ASIC | Application-Specific Integrated Circuit. |
| ATCA | Advanced Telecom Computing Architecture. |
| ATX | Advanced Technology Extended. |
| BER | Bit Error Rate. |
| CATV | Cable Television. |
| COTS | Commodity-off-the-shelf. |
| CPU | Central Processing Unit. |
| CRC | Cyclical Redundancy Check. |
| DC | Direct Current. |
| DDR | Double Data Rate. |
| DMA | Direct Memory Access. |
| DSP | Digital Signal Processor. |
| DVI | Digital Video Interface. |
| EDF | Earliest Deadline First. |
| EMI | Electromagnetic Interference. |
| f | Frequency In Hz. |
| FIFO | First In, First Out. |
| FPGA | Field Programmable Gate Array. |
| FLOPS | Floating-point Operations Per Second. |
| FR-4 | Flame Retardant Type 4. |
| HDM | High-density Metric. |
| I/O | Input / Output. |

| | |
|---|---|
| IBIS | Input Output Buffer Information Specification. |
| IC | Integrated Circuit. |
| IF | Intermediate Frequency. |
| IP | Internet Protocol. |
| JTAG | Joint Test Action Group. |
| LDO | Low Dropout. |
| LTI | Linear Time Invariant. |
| LVCMOS | Low Voltage Complementary Metal Oxide Semiconductor. |
| LVDS | Low Voltage Differential Signaling. |
| LVPECL | Low Voltage Positive Emitter Coupled Logic. |
| MAC | Multiply-accumulate. |
| MCU | Micro-controller Unit. |
| MGT | Multi-gigabit Transceiver. |
| mil | Milli-inch. |
| MIPS | Million Instructions Per Second |
| MTBF | Mean Time Before Failure. |
| NFET | Negative Channel Field Effect Transistor. |
| NTSC | National Television System Committee. |
| PCB | Printed Circuit Board. |
| PWM | Pulse Width Modulation. |
| RAM | Random Access Memory. |
| RF | Radio Frequency. |
| RGB | Red Green Blue. |
| RMS | Root Mean Square. |
| RJ-45 | Registered Jack 45. |
| S-param | Scattering Parameters. |
| SDR | Software-defined Radio. |
| smt | Surface Mount Technology. |
| SPICE | Simulation Program with Integrated Circuits Emphasis. |
| TAP | Test Access Port. |
| Tbps | Terabit Per Second. |
| TCP/IP | Transmission Control Protocol / Internet Protocol. |
| TV | Television. |

| | |
|---|---|
| uH | Microhenry. |
| USB | Universal Serial Bus. |
| VDC | Volts Direct Current. |
| VHDM | Very-high-density Metric. |
| XSVF | Xilinx Serial Vector Format. |

# Chapter 1

# Introduction

A modern class of problems – real time computation on live data – and the availability of high-performance, low-power commodity components (DSPs and FPGAs) has created an opportunity for new high-performance architectures and programming models. Reconfigurable computing responds to these problems by harnessing the bandwidth and flexibility of FPGAs. However, a computer that is organized entirely around FPGAs fails to leverage the recent availability of modern high-performance, low-power processors such as the Analog Devices Inc. TigerSHARC DSPs and the Texas Instruments TMS320C64 DSPs. By taking advantage of both of these things – the flexibility and bandwidth of FPGAs and the outstanding computational density of modern signal processors – we can design a supercomputing platform that delivers the benefits of both at fraction of the cost, in money and in power dissipation, of a system that relies exclusively on FPGAs such as Berkeley's BEE2 system [12].

The ILLIAC 6 project is an effort to develop a scalable, high-performance computing platform in hardware and in software using only commodity silicon. The machine is designed to operate on live streams of data in real-time at fixed bandwidths, such as for video or audio processing, packet processing, network security, or software defined radio. The system is organized around a reconfigurable module implemented as a mezzanine card that contains four powerful DSPs and an FPGA. The FPGA serves to interconnect modules in a bidirectional shuffle-exchange network, as an I/O interface, and as a reconfigurable computational resource

Traditional supercomputers like Blue Gene/L rely upon custom silicon and packaging to achieve great computational density [26]. However, full custom silicon markedly increases system development time and expense. Additionally, the I/O architectures and nondeterministic performance of these traditional supercomputers make them ill-suited for real-time computations on live data streams at high bandwidths. On the other hand, the ILLIAC 6 is designed from the ground up to support a deterministic performance model. All of the system interconnects have fixed latencies, and all of hardware and software components throughout the system support guaranteed bandwidths at the application level.

Furthermore, like many supercomputers, the ILLIAC 6 is designed for scaling to an enormous system with up to 65536 DSPs and 16384 FPGAs. However unlike most supercomputers, the ILLIAC 6 is also highly modular and designed to be powerful, inexpensive, and highly usable at smaller scales. A single module is a small and powerful computer that dissipates approximately 25 W. It can natively interface with numerous I/O protocols, and it offers 90 Gbps of I/O bandwidth. Furthermore, the single module programming model and tool chain apply directly to even the largest ILLIAC 6 configuration.

Finally, the programming framework lends itself to genuinely simple application development, because it utilizes a high-level block-diagram approach. This programming model borrows ideas from stream processing, macro dataflow, and real-time communication systems [34]. The model is *component-based*, where a component is a function that is ultimately mapped to the processors and/or FPGA fabric. Moreover, at the time of mapping, components can be configured to consume additional computational resources in exchange for additional performance. An application developer simply provides bandwidth requirements for the application as a whole, and interconnects the components to form a complete application. The program mapper manages the resources of the machine (computational resources, interconnect bandwidth, and input/output attachments) and dilates the components as needed to achieve the targeted performance.

In this thesis, our goal is to detail the architecture and the physical design of the ILLIAC 6 platform. In Chapter 2 we provide an overview of the system architecture and discuss some of the important overall design decisions. In Chapter 3 we present a detailed description of the smallest element in the system hierarchy, a module, and we discuss its role in providing enormous system scalability. In Chapter 4 we describe the carrier board design, and in Chapter 5 we overview the important engineering aspects of larger configurations including the chassis, subsystem, and system. In Chapter 6 we present and discuss our novel communications architecture and the FPGA's critical role in its design. In Chapter 7 we explain how the system is booted, configured, and debugged. In Chapter 8 we discuss the system programming model and applications, and some of the aspects of our hardware/software co-design. In Chapter 9 we consider related works and compare the relevant performance specifications. We conclude in Chapter 10 by summarizing our work and discussing how we plan to proceed in the future.

# Chapter 2

# Physical Design Overview

The ILLIAC 6 framework provides a highly scalable and flexible system architecture organized hierarchically at natural packaging boundaries as detailed in Table 2.1. The elementary building block is a module that contains four DSPs and an FPGA. The modules are intended to adhere to the emerging VITA 57 FPGA I/O Mezzanine card form factor. A module acts as a bidirectional 4x4 crossbar switch which can be connected to other modules to form a bidirectional shuffle-exchange network. Eight modules are mounted on a custom 8U carrier blade. Sixteen blades plug into a standard 23 in. telco enclosure, and two such enclosures form a chassis, which contains 256 modules. Even greater levels of scaling are provided for in the design: eight chassis can be interconnected to form a subsystem with 2,048 modules, and eight subsystems can be interconnected as a system with 16,384 modules. The hierarchical organization of the machine at each level causes the bisectional bandwidth and I/O bandwidth to scale together, and to scale nearly linearly with the number of modules. In this chapter we overview each level in the hierarchy.

The central idea underlying the physical organization of the machine is that each level of the system is designed to function as a bidirectional N-by-N crossbar and can therefore be used as the switch element in building the interconnect at the next highest level of the system. This is facilitated by the fact that at every level of the machine hierarchy, nodes are interconnected by a bidirectional full-shuffle exchange network. Within a module, the DSPs and the I/O channels are interconnected via the FPGA. In a carrier board, PCB traces connect modules together in a shuffle-exchange network. In a chassis, a custom midplane connects the carrier boards together to form a shuffle-exchange interconnection. In a subsystem and in a system, the shuffle-exchange networks are realized with high-quality copper cable assemblies. Table 2.2

| Level | DSP | FPGAs |
|---|---|---|
| Module | 4 | 1 |
| Carrier Board | 32 | 8 |
| Chassis | 1,024 | 256 |
| Subsystem | 8,192 | 2,048 |
| System | 65,536 | 16,384 |

**Table 2.1: Structural Hierarchy**

| Level | Interconnect Media | Crossbar Width |
|---|---|---|
| Module | FPGA | 4 x 4 |
| Carrier Board | PCB Traces | 16 x 16 |
| Chassis | Custom Midplane | 4 x 4 |
| Subsystem | Copper Cable Assemblies | 4 x 4 |
| System | Copper Cable Assemblies | N/A |

**Table 2.2: Interconnect Media and Topology**

shows this organization.

## 2.1 Module

The compute module is the cornerstone of the system. Almost all of the active components in the system are on the module, and the scalability of the system is made possible by the module architecture and implementation. By itself, a module is a powerful computer system with unprecedented I/O capabilities, but what makes the ILLIAC 6 module unique is the way in which it facilitates an enormous degree of seamless scaling to systems more than three orders of magnitude larger than a single module.

A module contains four Analog Devices Inc. TigerSHARC (ADSP-TS201) DSPs[3] and one Xilinx Virtex-2 Pro (XC2VP30) FPGA[54]. Each ADSP-TS201 is directly connected to 128 MB of SDRAM over an 8 Gbps memory interface, and each ADSP-TS201 is connected to the central FPGA via two bidirectional link ports with an aggregate communication bandwidth of 16 Gbps. Modules are interconnected via eight bidirectional serial MGTs [46] in the FPGA, providing 40 Gpbs of off-board bandwidth. Additionally, each module has a 50 Gbps interface exclusively for user I/O. The module contains an Atmel ATMega1280/V microcontroller[8] attached to the FPGA and to a Wiznet W3150A TCP/IP 10 Base-T ethernet interface [56]. This low-bandwidth network provides an out-of-band programming, monitoring, and debugging interface for the system. Figure 2.1 illustrates this module.

The DSPs are clocked at 500 MHz, and each DSP is capable of performing 4.8-billion MACs (16x16 with 40 bit accumulate) per second. The FPGA is also clocked at up to 500 MHz and more than 50% of the fabric is available for reconfigurable computation. (That is, less than 50% of the fabric is consumed by the architecture of the system itself.) The DSPs are connected to the FPGA via deterministic low-latency links. The transmit and receive buffers for these links are register-mapped within the processor. The links can be configured for DMA transfer as well, allowing the processor's internal memory to source or sink data without interrupting the processor core. A deterministic low-latency connection to and from the processor is a requirement for a large class of data intensive real-time applications, and is especially necessary for effectively using the FPGA as a coprocessor in such applications. With an approximate cost of $1000, a

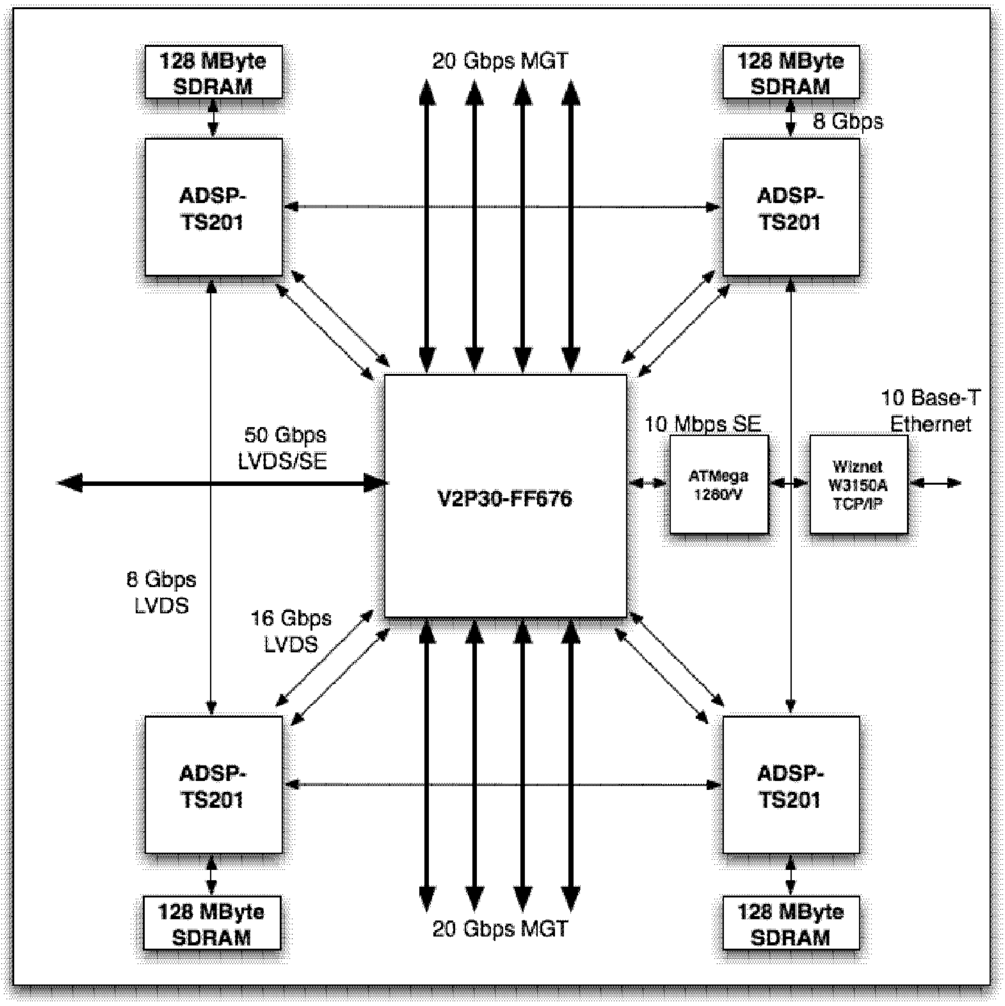

**Figure 2.1: Compute Module**

power budget of 25 W, and an I/O bandwidth of 90 Gbps, an ILLIAC 6 module is an extremely cost-effective and powerful reconfigurable computer.

The FPGA in a module performs both computational and communication functions. Eight bidirectional MGTs are used for inter-module communication. The interconnect of the ILLIAC 6 is bidirectional, and is partitioned into disjoint *up* and *down* networks. The networks are acyclic, making it straightforward to guarantee freedom from deadlock in message routing. For every bidirectional link in the system, one of the link-directions belongs to the up network and the other to the down network. The directions function entirely independently, except at the link layer (see Section 6) where they are used as a bonded pair to manage data retransmission for error recovery.

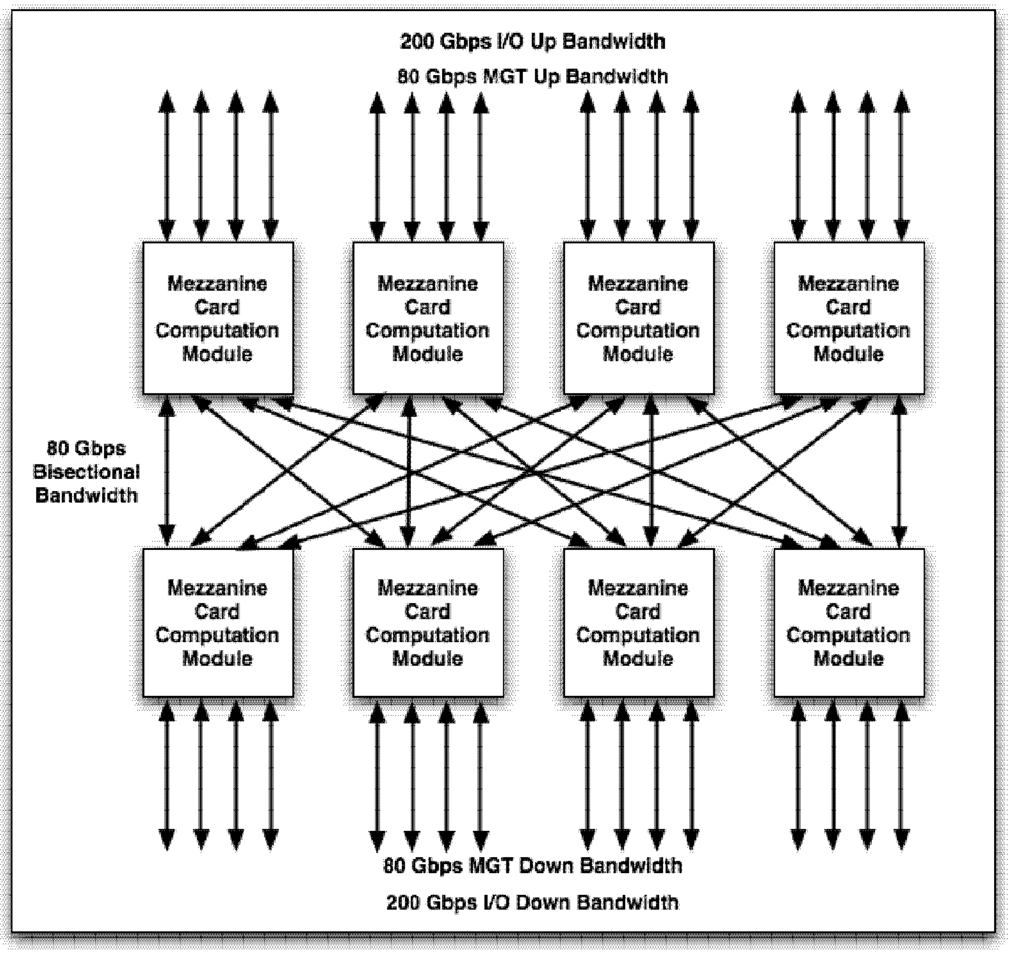
200 Gbps I/O Up Bandwidth
80 Gbps MGT Up Bandwidth
Mezzanine Card Computation Module
80 Gbps Bisectional Bandwidth
80 Gbps MGT Down Bandwidth
200 Gbps I/O Down Bandwidth

**Figure 2.2: Carrier Board (2.5 Gbps per link direction)**

As shown in Figure 2.1, the upward-pointing halves of the MGTs belong to the up network, and the downward-pointing halves belong to the down network of the machine. Each MGT has a maximum bandwidth of 2.5 Gbps in each direction. The MGTs can be configured to work with a number of different protocols. This flexibility allows any module to act as either an internal node with both sets of MGTs connecting it to the system interconnect, or as an ingress or egress point for system I/O. Most I/O protocols are directly supported by the MGTs and a connection can be achieved with a simple electrical and mechanical form factor conversion board. Any I/O type that is not directly supported can be attached to the 50 Gbps I/O port. When configured for inter-module interconnection, the MGTs utilize the Xilinx 8B/10B Aurora Protocol[35].

The 50 Gbps I/O port (as seen in Figure 2.1), is realized with a 144-pin HDM connector. 64 general

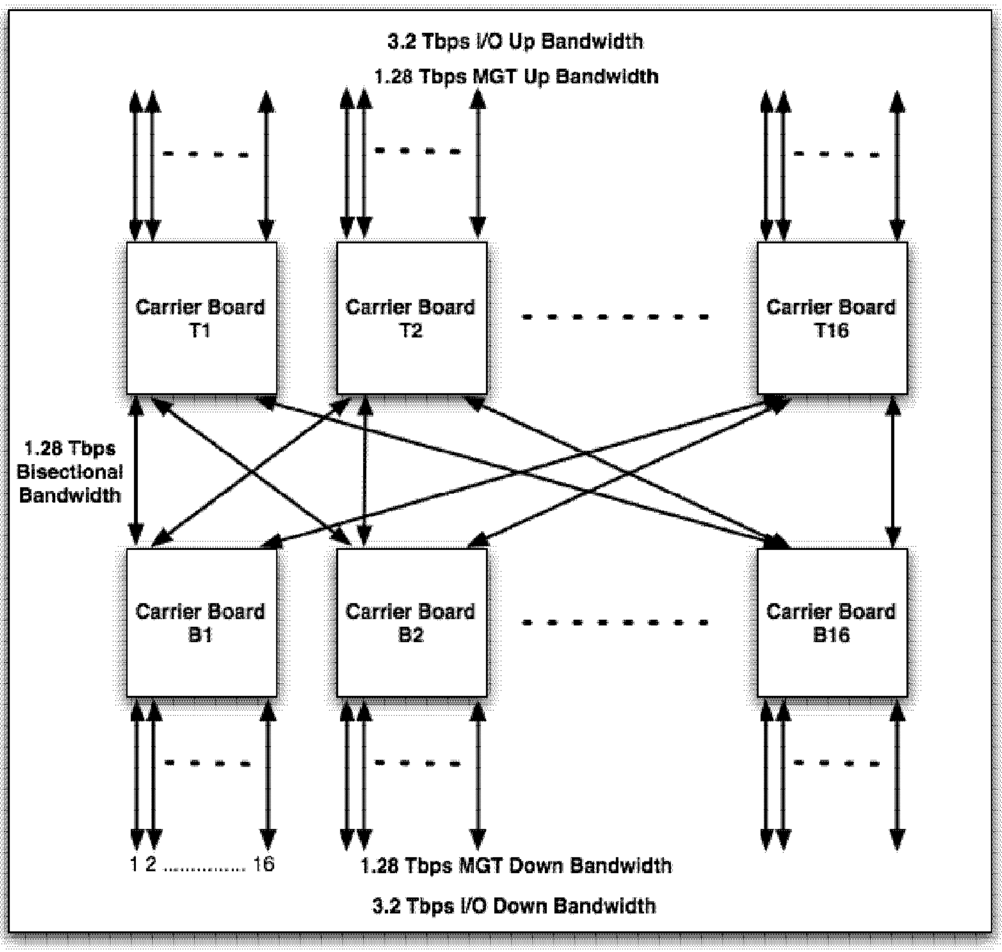
3.2 Tbps I/O Up Bandwidth
1.28 Tbps MGT Up Bandwidth
Carrier Board T1
Carrier Board T2
Carrier Board T16
1.28 Tbps Bisectional Bandwidth
Carrier Board B1
Carrier Board B2
Carrier Board B16
1 2 ............... 16
1.28 Tbps MGT Down Bandwidth
3.2 Tbps I/O Down Bandwidth

**Figure 2.3: Chassis (2.5 Gbps per link direction)**

purpose I/O pairs are routed from the FPGA to this connector. These pairs are user configurable as input, output, or both, they support most 1.5 V, 1.8 V, 2.5 V, and 3.3 V electrical protocols, and they can provide 50 Ω or 100 Ω source or end termination via soft configuration. The full list of directly supported protocols can be found in [54] and [53]. An I/O bandwidth of 50 Gbps can easily be achieved by configuring all 64 pairs as 500 MHz DDR LVDS I/O, and using 50 pairs for data, 6 pairs as differential clocks, and 8 pairs for flow control.

## 2.2 Carrier Board

A carrier board holds eight mezzanine cards organized as a bidirectional shuffle-exchange network of modules. The carrier board dissipates approximately 230 W powering eight 25 W mezzanine cards. Each module

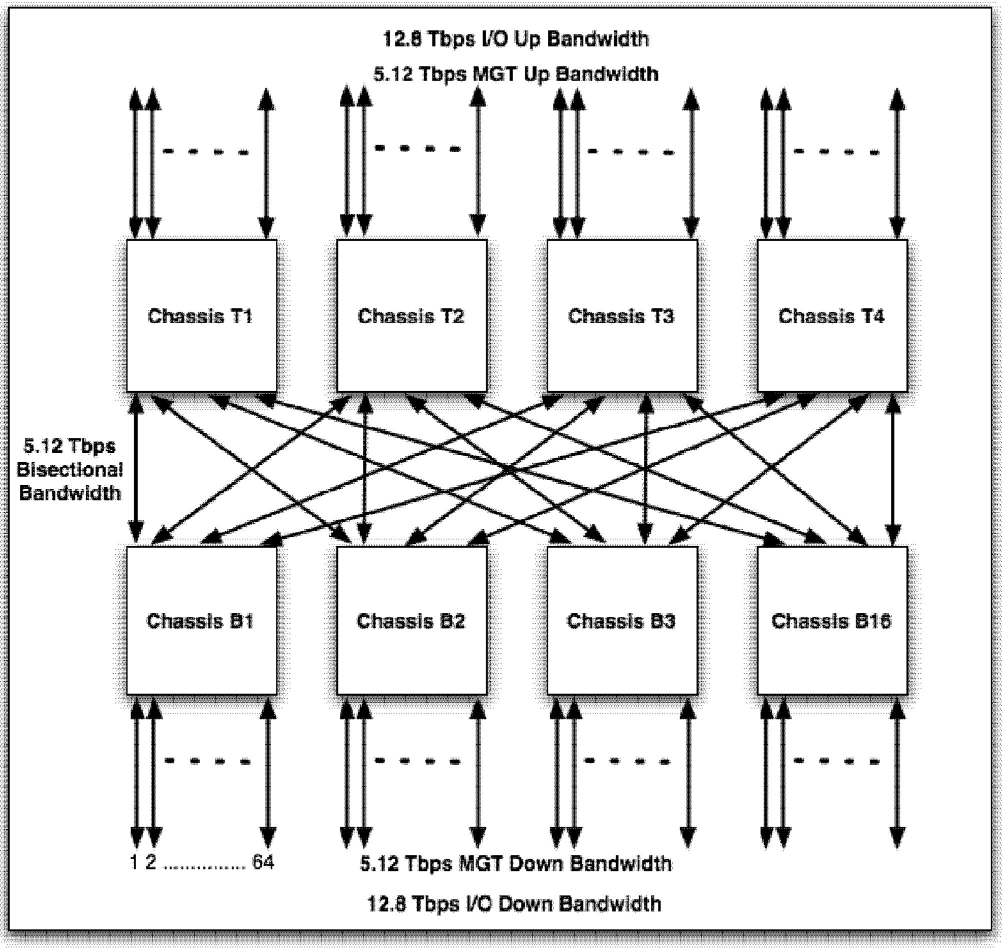
12.8 Tbps I/O Up Bandwidth
5.12 Tbps MGT Up Bandwidth
Chassis T1
Chassis T2
Chassis T3
Chassis T4
5.12 Tbps Bisectional Bandwidth
Chassis B1
Chassis B2
Chassis B3
Chassis B16
1 2 .............. 64
5.12 Tbps MGT Down Bandwidth
12.8 Tbps I/O Down Bandwidth

**Figure 2.4: Subsystem (2.5 Gbps per link direction)**

contains eight FPGAs, 32 DSPs, and 4 GB of RAM, and costs less than $10000. Acting as an internal node, each populated board has an input bandwidth of 80 Gbps, an output bandwidth of 80 Gbps, and a bisectional bandwidth of 80 Gbps. Acting as an ingress node, a board has an input bandwidth of 280 Gbps and an output bandwidth of 80 Gbps; acting as an egress node, it has an input bandwidth of 80 Gbps and an output bandwidth of 280 Gbps. Figure 2.2 diagrams the board.

## 2.3 Chassis

A chassis consists of two 23 in. rack-mounted enclosures connected by a custom passive mid-plane. Each enclosure houses 16 carrier boards for a total of 32 carrier boards in the chassis. The boards are connected in a full shuffle-exchange network, and in the context of a chassis, a carrier board can be viewed as a full 16x16

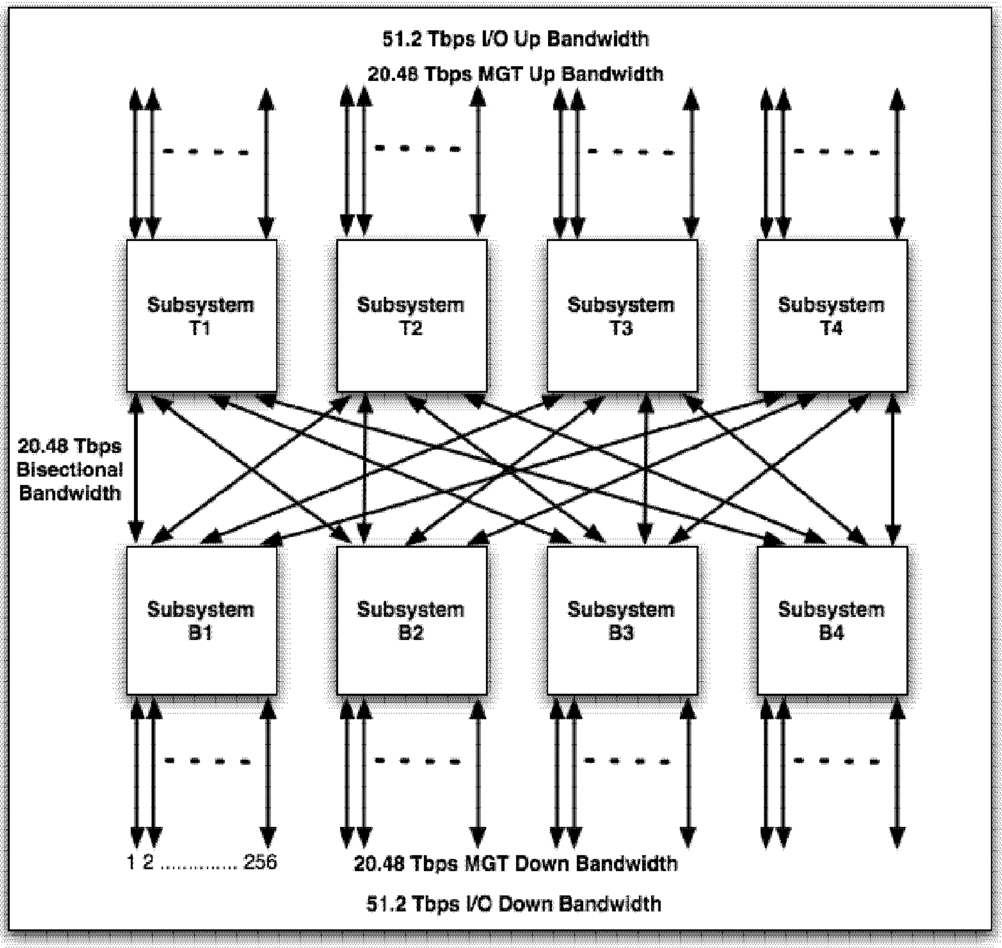
51.2 Tbps I/O Up Bandwidth
20.48 Tbps MGT Up Bandwidth
Subsystem T1
Subsystem T2
Subsystem T3
Subsystem T4
20.48 Tbps Bisectional Bandwidth
Subsystem B1
Subsystem B2
Subsystem B3
Subsystem B4
1 2 ............ 256
20.48 Tbps MGT Down Bandwidth
51.2 Tbps I/O Down Bandwidth

**Figure 2.5: Full System (2.5 Gbps per link direction)**

crossbar switch. A chassis internal to the system has a 1.28 Tbps input bandwidth, a 1.28 Tbps output bandwidth, and a 1.28 Tbps bisectional bandwidth. Acting as an ingress point to the system, the input bandwidth of a chassis is 4.48 Tbps, and acting as an egress point to the system, the output bandwidth is 4.48 Tbps. Each chassis is designed to dissipate approximately 8 KW of power, and populated contains 256 FPGAs, 1024 DSPs, and 128 GB of RAM. Figure 2.3 depicts a chassis.

## 2.4 Subsystem

As the system scales, any simple aggregation of the connections is possible, so a chassis can function as an 8x8 crossbar, or in the context of a subsystem as a 4x4 crossbar of aggregated links. A subsystem has 5.12 Tbps of input bandwidth, 5.12 Tbps of output bandwidth, and 5.12 Tbps of bisectional bandwidth.

It contains 2,048 FPGAs, 8,192 DSPs, and 1 TB of RAM, and dissipates approximately 64 KW of power. When acting as an ingress or egress point to the system, the input and output bandwidth are 17.92 Tbbps respectively. Figure 2.4 diagrams a chassis.

## 2.5 System

In the context of a system, a subsystem forms a 4x4 crossbar switch of aggregated links. A full system has an aggregate I/O bandwidth of 92.16 Tbps and a 20.48 Tbps bisectional bandwidth. It contains 16,384 FPGAs, 65,536 DSPs, and 4 TB of RAM. It dissipates approximately 500 KW of power. Figure 2.5 diagrams a full system.

## 2.6 Upgrade Path

An important aspect of leveraging COTS components in our system is the provision of a seamless drop-in upgrade path for modules using newer and faster components. When higher-speed Virtex series FPGAs, such as the Virtex-4 or Virtex-5, become available, we will design new modules with the same form factor, but with much higher-speed I/O. The Virtex-4 FPGA promises 10 Gbps MGT I/O [51], a factor of four improvement, and preliminary information on the ADSP-TS201 successor, the ADSP-TS301, indicates a twofold improvement in link-port speed. However, in order to make drop-in module upgrades possible, the carrier board, chassis, and interconnects must be architected and designed to scale with improved component capabilities. The most difficult aspect of achieving this high degree of scaling is ensuring adequate signal integrity on all signal paths for much higher-speed I/O than utilized in the initial system.

# Chapter 3

# Module Design

A module is implemented as a custom, 18-layer, low-cost FR-4 PCB in a mezzanine card form factor as depicted in Figure 3.1. The high-speed MGT signals leave the module via two VHDM connectors located on the far left and far right ends of the board. The 50 Gbps I/O connections are made via the HDM connector on the right side of the board. The power connector is a standard 4-pin Molex connector, and the ethernet interface is implemented with a standard RJ-45 connector. The ethernet interface is also routed to the VHDM connectors on the right side of the board. The VHDM connectors can mate to either a cable assembly or to the complementary connector on a carrier board.

The most challenging aspects of the board design are power regulation, ensuring proper clock distribution, and guaranteeing signal integrity for all nets. Providing adequate power to all components with minimal dropout and ripple is a necessity for proper operation of high-speed ICs [55], and the large current demands of the components make this a difficult task. Providing low-skew and low-jitter clocks with a high degree of crosstalk immunity is also critically important for high-speed ICs [55]. Finally, high-speed signaling has been and continues to be an extremely important and active area of research, and it necessitates careful board design and simulation [29], [30], [55]. This chapter details each of these aspects of the module design.

## 3.1 Power

The FPGA and the DSPs require four different electrical potentials to ground, realized as four separate power nets, in order to operate, and the other active components use only these four nets. The maximum possible current drawn on each net sets a lower bound for the amount of current that the components in each power net must be able to tolerate, and it acts as a key parameter in designing and implementing the regulation circuits. Table 3.1 provides these current bounds.

An external 12 VDC source is used to power the boards; using a 12 V power supply for the modules is a good compromise over using a 24 V or 48 V source, and it simplifies a number of aspects of the design. For small systems, a simple low-cost ATX power supply can be utilized as a power source. Secondly, it allows a

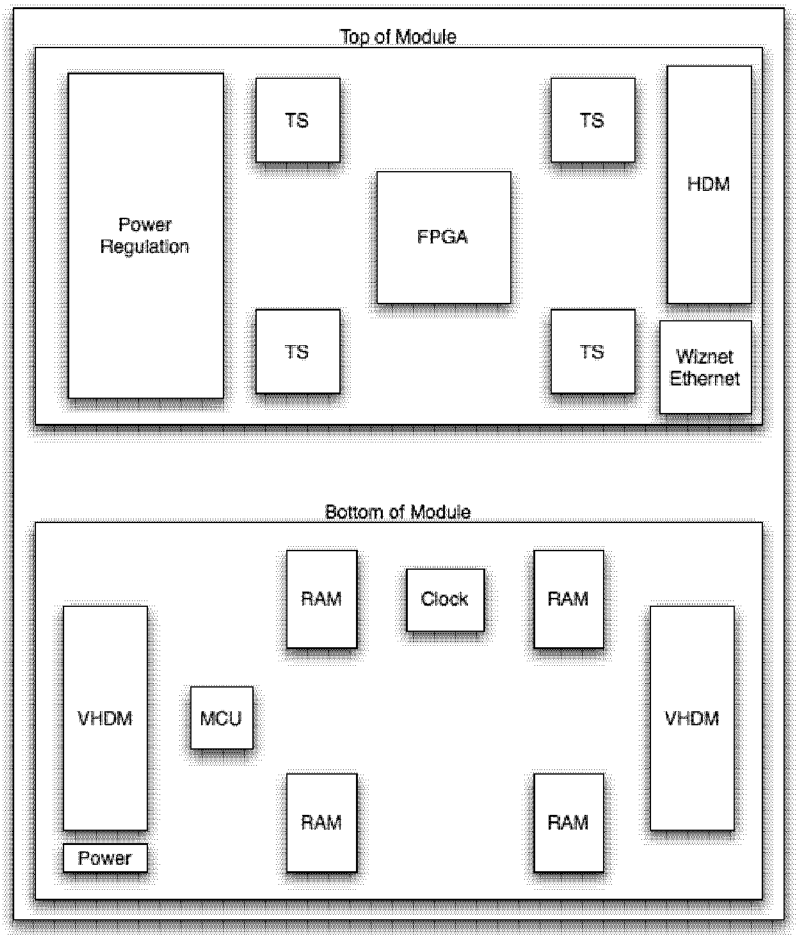

**Figure 3.1: Module Layout**

much larger and richer set of power regulation parts to be used, because fewer regulation ICs are rated for 24 V or 48 V than for 12 V. Finally, using a 12 V source reduces the maximum RMS input current ripple, and it allows for a faster switching frequency to be used, which in turn reduces the size and cost of the output capacitors and inductors. However, using a 12 V input clearly results in a larger average current draw than using a higher voltage source, assuming similar conversion efficiencies. This makes power distribution over a carrier card slightly more difficult.

In order to maximize the power conversion efficiency, the four power domains are generated and regulated with a high-efficiency dual NFET buck switching regulation circuit as seen in Figure 3.2. A simplified version of a buck regulator circuit is depicted in Figure 3.3. The circuit derives $V_{out}$ from $V_{in}$ by using a form of PWM regulation. The PWM controller provides a continuous pulse train to the gates of the NFETs, which

| Net (V) | Max Current (A) |
|---|---|
| 1.05 | 14.20 |
| 1.50 | 5.60 |
| 2.5 | 3.25 |
| 3.3 | 2.93 |

**Table 3.1: Power Networks Maximum Current**

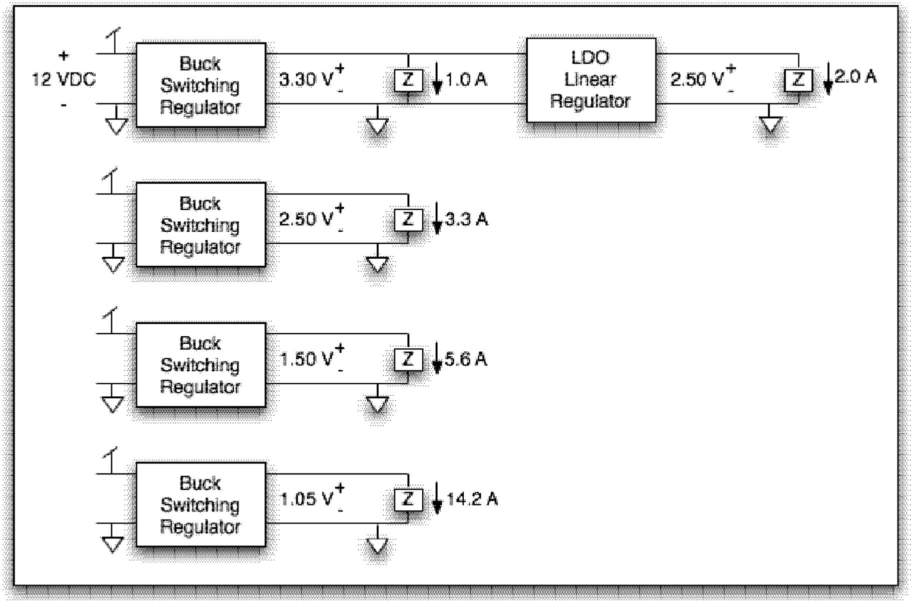

**Figure 3.2: Architectural View of Power Regulation**

switch on and off at the rate of the PWM duty cycle. The inductor and the capacitor on the output net act as an LC low-pass filter and convert the pulse train into the steady DC voltage, $V_{out}$. A feedback loop from $V_{out}$ to the PWM controller allows the controller to dynamically adjust the duty cycle in order to provide an accurate $V_{out}$. The disadvantage of this regulation scheme is that the peak input current is equal to the peak output current, even though the average current is reduced by $V_{out}/V_{in}$ [55]. The solution to this peak current problem is to use a flyback regulator as depicted in Figure 3.4, which is a buck regulator that isolates the inductor from the output capacitor via a transformer. A transformer with N:1 turns scales the peak input current down by a factor of N. Unfortunately, this solution is impractical for a module, because transformers that are able to handle the peak currents in the module power nets are simply too large and bulky to be used on a small mezzanine card.

In order to minimize the total RMS input current, we used two dual-phase synchronous PWM controllers to generate the four required voltages [36]. These controllers are able to regulate two different channels,

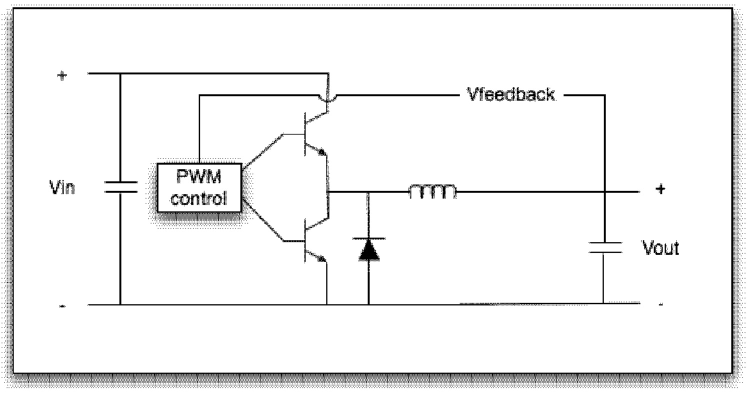

**Figure 3.3: Simple Buck Regulator Circuit**

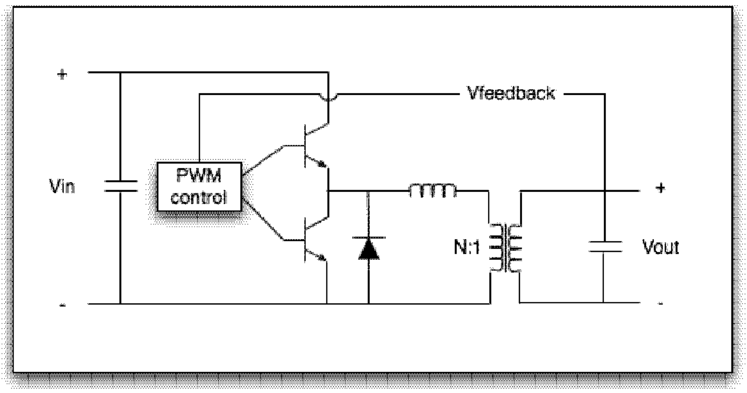

**Figure 3.4: Simple Flyback Regulator Circuit**

which are modulated 180° out of phase. By forcing the controllers to synchronize to an external clock, and by providing a delayed copy of the clock to one of the controllers, we are able to generate four non-overlapping phases of operation. This in effect interleaves the current pulses drawn by the switches, and provides a significant reduction in the total RMS input current. This in turn greatly reduces the size and ESR constraints of the input capacitors and reduces their power losses; it also results in much lower EMI [36].

The external clock frequency determines the switching frequency of the controllers and is a very important design parameter as it directly effects the sizing of the output inductor and capacitors. The ripple current through the inductor is given by Figure 3.5. The output voltage ripple is directly proportional to the current ripple, so obtaining a low current ripple is essential to obtaining a low voltage ripple. It is clear from Figure

$$\Delta I_L = \frac{V_{Out}}{fL}\left(1 - \frac{V_{Out}}{V_{In}}\right)$$

**Figure 3.5: Inductor Ripple Current**

3.5 that the current ripple is inversely proportional to both the switching frequency, $f$, and the inductor value, $L$, so using a high switching frequency is advantageous as it allows for a smaller inductor to be used without increasing the current ripple. High amperage inductors in the $\mu$H to mH range are both expensive and large, so using the highest practical switching frequency is a good design decision. However, high-current NFETs cannot switch instantaneously, and every controller has a minimum on-time. The lowest $V_{out}$ and the minimum on-time of the control set the upper bound for the switching frequency. In a module, the maximum safe switching frequency is 270 KHz, and using a 500 ns delay line on the clock for the 3.3 V and 2.5 V domains results in the timing depicted in Figure 3.6

Finally, the power supply sequencing and ramp-up times must be controlled to prevent permanent damage to the silicon devices due to over-current conditions. The Virtex-2 Pro has a ramp-up time requirement T for $V_{CCINT}$: 200 us < T < 50 ms, and it has the sequencing constraint that $V_{CCAUX}$ powers up before or at the same time as $V_{CCO}$ [54]. The ramp-up time is guaranteed at the controller by using capacitors to ensure that all supplies ramp up in under 50 ms, and the sequencing constraint is satisified by using the same power plane for $V_{CCAUX}$ and $V_{CCO}$. The ADSP-TS201 has no ramp-up time requirements, but the processor must be held in the reset state until all power rails are stable; it has a sequencing constraint that $V_{DD_{RAM}}$ must power on last [21]. To keep the processors in the reset state during power up, weak pull-downs are placed between \RST_IN and ground. To ensure the ramp-up constraint, the 1.50 V network, which is connected to $V_{DD_RAM}$, is delayed 50 ms, and takes 50 ms to ramp up. All of the other nets are delayed 12.5 ms and ramp-up in 12.5 ms.

Ultimately, we were able to design all of the power nets to have less than a 0.5% voltage ripple, without using overly expensive or large components. However, in order to provide an even cleaner supply for the MGTs, we utilized a 2.5 V LDO linear regulator powered off of the 3.3 V net as depicted in Figure 3.2. Lastly, since each power net's $V_{out}$ is generated at the node formed by the output capacitor and inductor on the top side of the board, sufficiently wide traces and vias must be utilized to connect this node to the internal power planes. This is an important aspect of the power supply design, because the maximal current demands on the 1.05 V and the 1.50 V nets can result in substantial heat generation, and the inductance of the vias can become problematic.

The design rule IPC-2221 for the current carrying capacity of traces on printed circuit boards is widely used, but it is overly conservative and uses a simple and dated board model. For our design, we use the

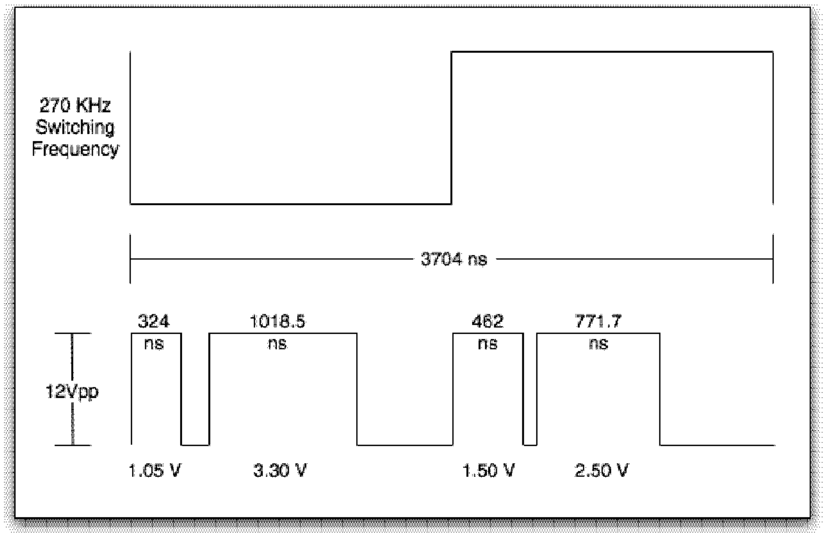

**Figure 3.6: Switching Regulation Timing Diagram**

much more modern models published by Johannes Adam in SEMI-THERM: 2004 [2]. For vias, we make the assumption that via wall thickness is approximately the same as trace thickness, and then we use the cross sectional area of conduction to generate an equivalent trace width, which allows us to use the same thermal model as for traces. The trace width and via requirements for a module's power nets are shown in Table 3.2.

Lastly, in order to manage the heat dissipation of the NFETs and the linear regulator, we leverage the numerous ground planes, with their high thermal conductivity. The NFETs and the linear regulator are smt devices, and we have modified their footprints by adding a small copper ground plane on the top layer between the two rows of pads. Each of these planes is connected by thermal vias to every ground plane in the stackup, and the packages are in direct contact with top layer plane, so a great deal of heat will be dissipated directly into the PCB. The FPGA and the DSPs utilize heat sinks for thermal relief, and the MCU and RAM produce very little heat, so using the PCB planes to cool the power network is an adequate solution. In the context of a larger system, such as a chassis, thermal relief via heat conduction to the chassis and convection cooling via fans is a major engineering challenge and an essential part of the system design, but it does not greatly influence the design of thermal relief at the module level.

## 3.2 Clocks

In order to reduce power consumption on a computationally underutilized module, it is important to be able to selectively reduce the clock frequencies of the DSPs, the FPGA fabric, and the MGTs. We are able to

| Net (V) | Max Current (A) | Min Trace Width (mil) | Min Via Count |
|---|---|---|---|
| 1.05 | 14.20 | 315 | 6 |
| 1.50 | 5.60 | 79 | 3 |
| 2.5 | 3.25 | 39 | 2 |
| 3.3 | 2.93 | 39 | 2 |

**Table 3.2: Power Net Required Trace Widths on Top Layer Using 2 oz Copper and Required Via Count Using 10 mil Vias to Power Planes With a Maximum $\Delta T$=20°C**

determine the minimum required clock rates to achieve a minimal bandwidth at compile time, and these clock frequencies become part of the program binaries. A module supports this model with several separate programmable clock nets generated with the MAX9492 clock generator IC [40]. The clocks are buffered and distributed to the DSPs, RAM, and FPGA fabric using the Cypress CY23EP05 5-output zero-delay buffer [16]. The RMS jitter for each of the MAX9492 outputs is less than 10 ps, and the RMS jitter for each of the CY23EP05 drivers is less than 13 ps, so the total expected clock jitter is extremely low. The MGT clocks and FPGA fabric for MGT interfacing are also generated with the MAX9492, but the MGTs require either an LVDS or an LVECPL clock signal, so two MAX9372 [39] LVTTL to LVPECL level translators are used to provide the MGT related cocks. The MAX9372 adds fewer than 40 ps of peak-to-peak jitter and has less than 10 ps output to output skew. Finally, the output loads and the transmission line lengths for each of the local nets are closely matched to minimize distribution skew and jitter. The clock distribution networks is shown in Figure 3.7. The Atmel MCU is clocked with its internal oscillator, and a 25 MHz crystal clocks the Wiznet TCP/IP chip.

All single-ended clock traces are implemented with 50 Ω transmission lines and are source-terminated. The single-ended traces are also bordered with guard traces to reduce crosstalk. The LVPECL clock traces are implemented with 100 Ω differential transmission lines and are end-terminated with a split termination network biased to the LVPECL switching threshold.

## 3.3 Signal Integrity

High-speed digital systems engineering takes into account all of the passive components that are generally ignored in low-speed systems, such as traces, PCB dielectrics, packages, vias, and solder points, because at high-speeds, these passive components greatly affect edge rates, propagation velocity, and overall signal integrity [55]. Furthermore, high-speed designs account for both capacitive and inductive coupling between traces, which results in signal cross-talk and impedance variations. Ultimately, high-speed digital systems need to be engineered from the ground up to ensure a high-degree of signal integrity, and through careful

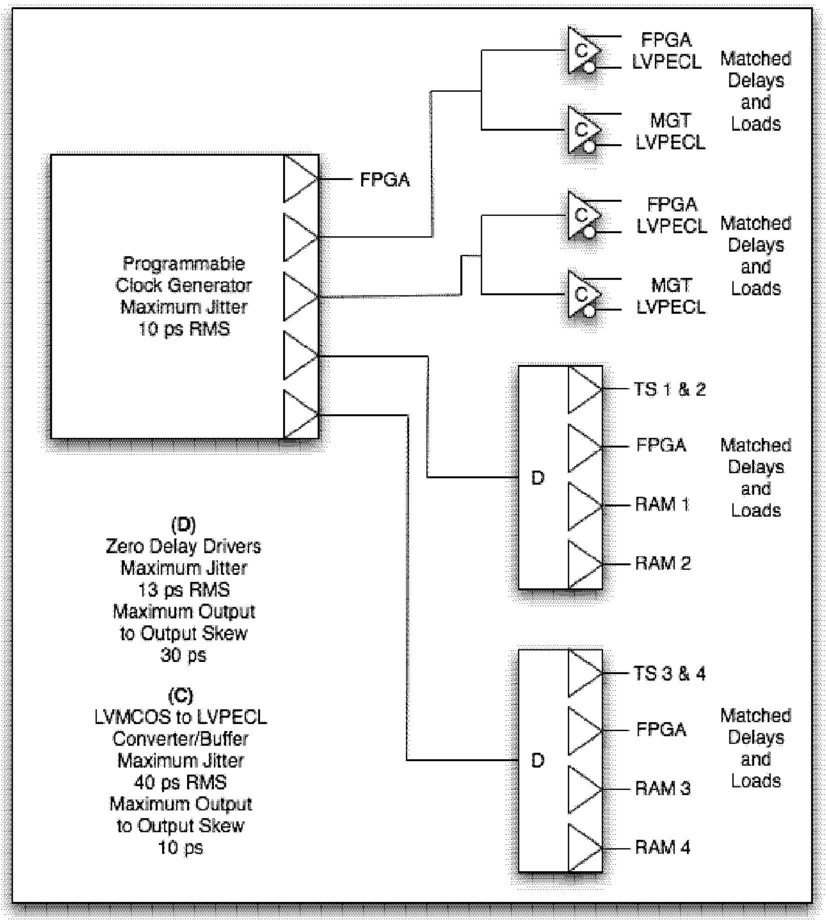
Programmable Clock Generator Maximum Jitter 10 ps RMS
FPGA
C
FPGA LVPECL
MGT LVPECL
Matched Delays and Loads
FPGA LVPECL
MGT LVPECL
Matched Delays and Loads
D
TS 1 & 2
FPGA
RAM 1
RAM 2
Matched Delays and Loads
TS 3 & 4
FPGA
RAM 3
RAM 4
Matched Delays and Loads
(D) Zero Delay Drivers Maximum Jitter 13 ps RMS Maximum Output to Output Skew 30 ps
(C) LVMCOS to LVPECL Converter/Buffer Maximum Jitter 40 ps RMS Maximum Output to Output Skew 10 ps

**Figure 3.7: Clock Distribution Network**

design and simulation of all signal paths, worst case BERs can be calculated before boards are manufactured and assembled.

### 3.3.1 PCB

Every module's components, from the simple passive resistors and capacitors to the 676-pin FPGA, are soldered to and interconnected on a printed circuit board. Choices of package types, routing constraints, and board floor planning play an important role in ensuring signal integrity, but it is ultimately the physical parameters of the PCB that set the upper bound on signal quality. Nearly every trace in our design utilizes transmission lines as opposed to simple point-to-point traces. However, as in all designs, these transmission lines are inherently lossy, and they suffer skin effect and dielectric losses at high frequencies. With use of the cheapest board materials – FR-4 and copper – these losses can be substantial. The losses can be greatly reduced using more exotic materials like ceramic and gold, but the price of multilayer FR-4 boards is already appreciable, and using even a slightly better dielectric such as GETEK with copper greatly increases the board manufacturing cost. In the ILLIAC 6, our careful design and layout allows us to utilize the most inexpensive PCB materials while maintaining good signal integrity at serial speeds of up to 10 Gbps, as indicated by simulation.

All signals in our design utilize either microstrip or stripline transmission lines in order to reduce distortion, EMI, and crosstalk [29]. Furthermore, in order to minimize transmission line impedance variation and crosstalk, we use a conservative custom layer stack detailed in Section 3.8. In this 18-layer FR-4 stack, all stripline transmission lines, except for the two central routing layers, are sandwiched between a ground and a power plane. The two innermost layers are separated by a thick core and are routed orthogonally to each other. This stackup increases the number of layers required to route the signals, but it greatly reduces the parasitic effects of neighboring traces at the first order to only directly adjacent traces in the same plane. In a more traditional stackup utilizing vertical and horizontal routing on neighboring planes, all of the traces in adjoining planes in addition to directly adjacent traces in the same plane have appreciable parasitic affects.

### 3.3.2 Simulations and Eyes

Even with a conservative layer stackup and conservative routing constraints, performing detailed electrical simulations of the PCB is essential to ensuring proper signal integrity. We utilize top-of-the-line simulation tool suites including HyperLynx, HSPICE, and MATLAB to carefully model and simulate all electrical aspects of the design. As of the creation of this document, the PCB routing is not yet complete, so all simulations in this document are performed assuming worst case routing distances. When the layout is complete, we will be able to simulate the final board in order to detect and correct both impedance variations and unexpected crosstalk.

All channel models include SPICE driver and receiver models when available, but IBIS models are used when SPICE models are not available. Package parasitics and connectors are modeled with S-params, and vias are modeled using simple but accurate LC circuits. We use HyperLynx for its 2D field simulation and hyper-accurate lossy transmission line models. Most simulated channels are passive LTI systems, making frequency-domain simulation and eye diagram generation possible. The off-board MGT channels discussed in Section 4.3 utilize active equalization circuits that are neither passive nor linear, so the corresponding simulations are performed in the time domain.

The most important high-speed module component interfaces are the TigerSHARC to RAM connections, the TigerSHARC to Virtex-2 Pro link-ports, and the Virtex-2 Pro MGT to MGT links. We have performed detailed simulations of all of these channels and provide eye diagrams for the first two in Figures 3.9 and 3.10. The full MGT channels simulations are discussed and eyes displayed in Sections 4.3 and 5.4

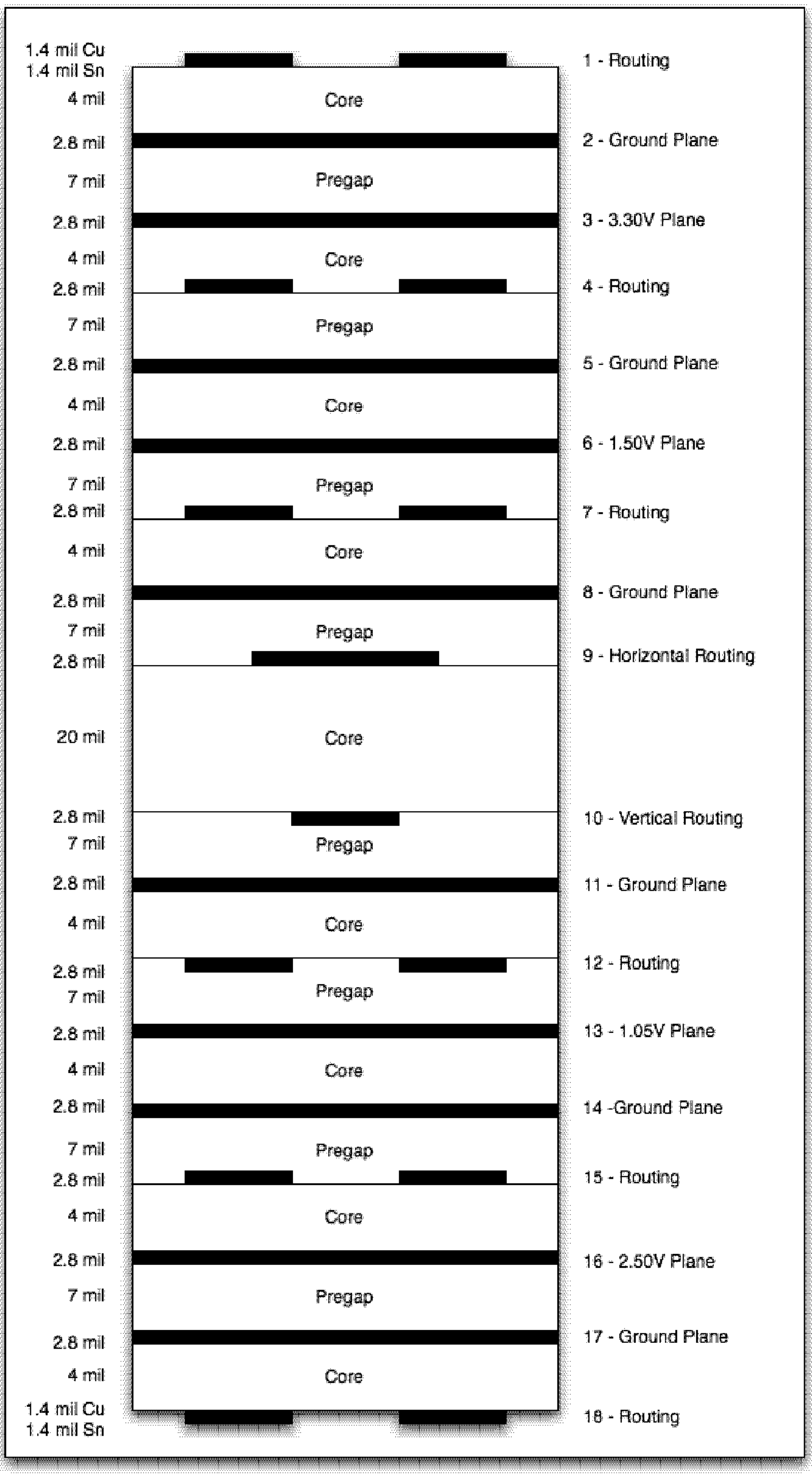

Figure 3.8: Module Layer Stackup

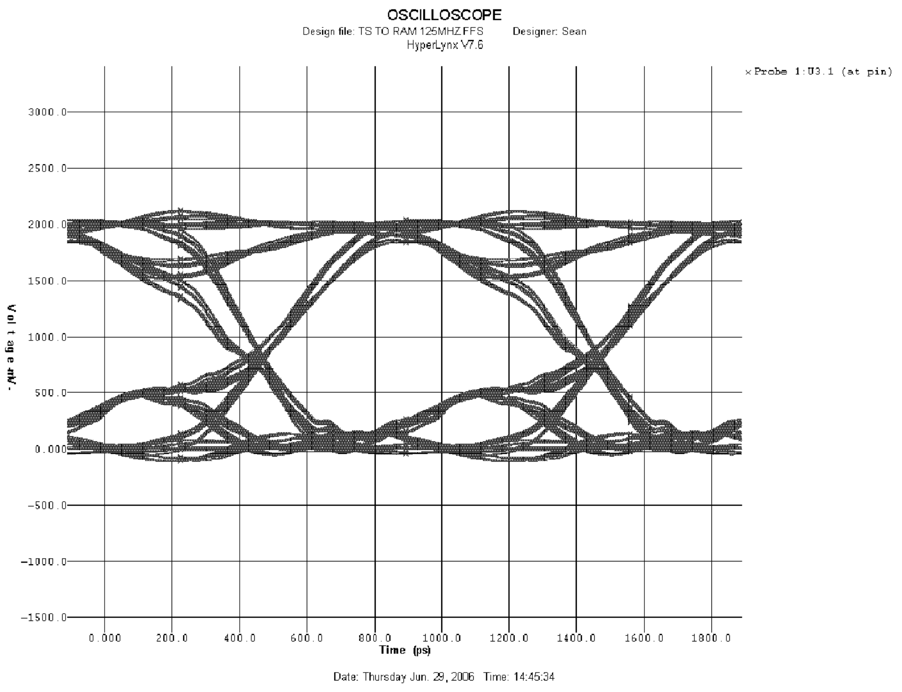

Figure 3.9: Eye Diagram TigerSHARC to SDRAM Data Channel

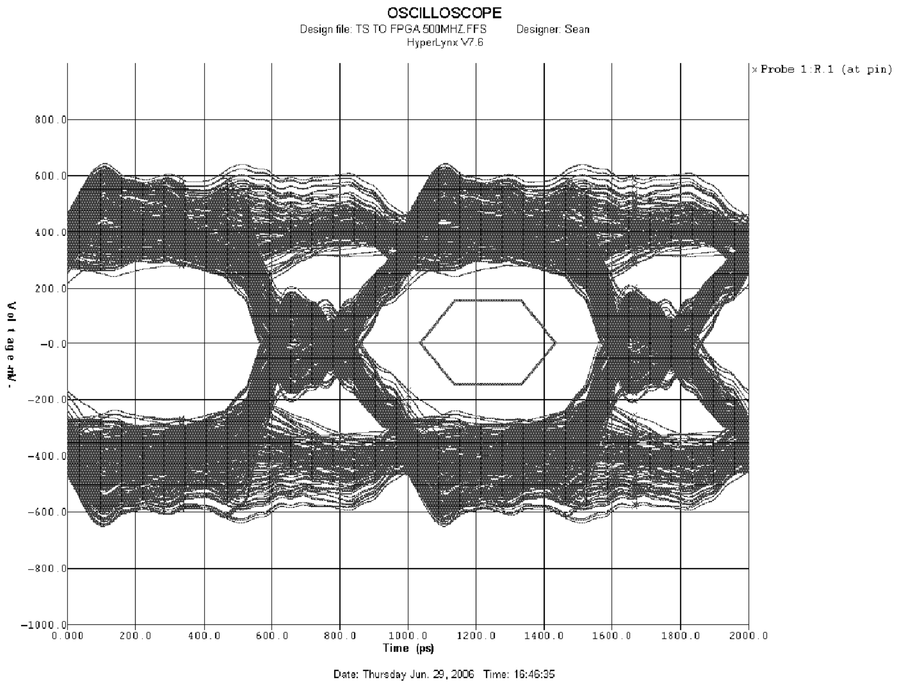

Figure 3.10: Eye Diagram TigerSHARC to Virtex-2 Pro Link-Port Data Channel

# Chapter 4

# Carrier Board Design

Much like the mezzanine card, the carrier board can be configured as a full computer with I/O on both edges, as an ingress or egress point, or as a component within a larger system. As a full system, a populated carrier board is a powerful computer. It contains 8 FPGAs, 32 DSPs, and 4 GB of RAM, and it dissipates approximately 230 W for a total cost of less than $10,000. In this chapter, we provide an overview of the carrier board architecture and layout, and we detail the most important aspects of the design: power distribution, and signal integrity.

## 4.1 Carrier Board Overview

A carrier board is implemented as a custom, 10-layer, low-cost FR-4 PCB as depicted in Figure 4.1. The carrier board holds 8 mezzanine card compute modules connected via VHDM signal and Molex power connectors; on the carrier board, the 8 modules are configured as 2 rows of 4 columns as shown in Figure 4.2. The two rows of modules are connected by a 16x16 full-shuffle exchange network implemented as passive components on the carrier board. I/O signals enter and leave the carrier board via one row of VHDM connectors on the top edge of the board and another row on the bottom edge of the board. System I/O utilizing the 50 Gbps I/O headers bypasses the carrier board and plugs directly into the mezzanine cards on the bottom row. The ethernet debugging and programming interfaces are also routed out to the top edge VHDM connectors. Finally, the boards are powered via Molex power connectors located beside the top-edge VHDM connectors.

The 16x16 full-shuffle exchange is implemented passively with stripline transmission lines. This shuffle-exchange network is the realization of the architecture discussed in Section 2.2 and shown in Figure 2.2. As discussed in Section 5, the top edge of the board can mate with our custom midplane used within a chassis. This is facilitated by the fact that the VHDM connectors used on the board can mate to either a cable assembly or to the complementary connector on the midplane. The bottom edge of the carrier board may connect to system I/O or to other carrier boards in other chassis, and in the context of an internal node

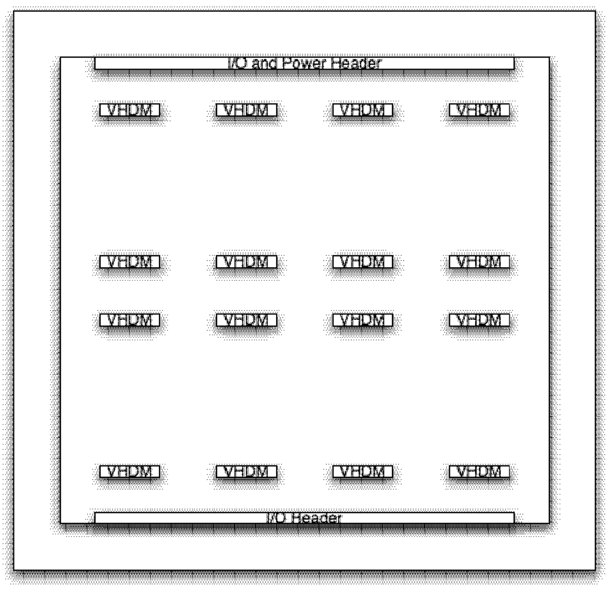

**Figure 4.1: Unpopulated Carrier Board Layout**

this connection may require up to 30 ft of cabling. This lengthy run necessitates the use of active signal equalization and amplification to achieve the targeted bandwidth, so all incoming links on the bottom edge of the board include active equalization circuitry.

## 4.2 Power

The carrier board is responsible for converting the -48 VDC rails provided by the chassis and midplane to a 12 VDC domain. To power the carrier board, a small power controller circuit collaborates with the midplane power manager as detailed in Section 5.2. As discussed in Section 3.1, each module requires a 12 V power source and has an average a power consumption of only 25 W. However, due to the choice of power regulation circuitry, the maximal transient current draw may be as high as 14.2 A. With 8 modules on a carrier board, at the worst case, a transient current draw of up to 113.6 A is possible. By using the flyback power regulation circuitry detailed in Section 3.1 it is possible to reduce this maximum current demand by a factor of four to 28.4 A in the -48 V domain.

Using a 2 oz, 12 V power plane and numerous high current Molex connectors makes distribution feasible, but the possible 113.6 A transient current demand necessitates a number of large low ESR power capacitors in addition to a GND - FR-4 - VCC capacitive sandwich in the PCB stackup. Furthermore, from the trace current-carrying capacity models in [2], it is clear that with 2 oz copper layers at least 4 separate power

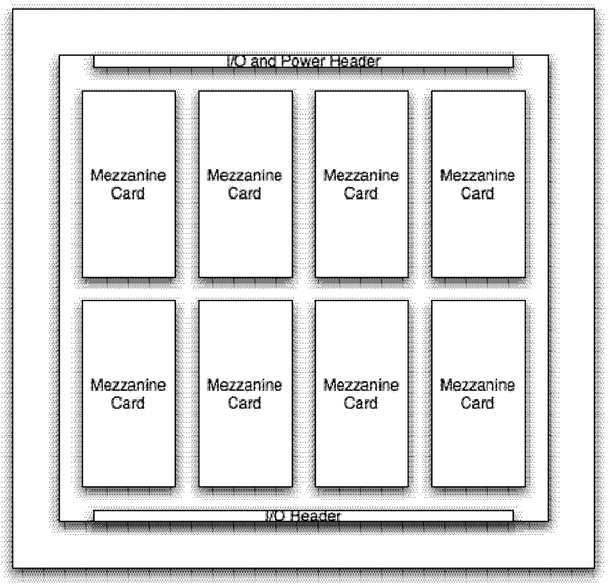

**Figure 4.2: Populated Carrier Board**

connectors and capacitor nets are required in order to keep the power-carrying trace widths less than 1 in. wide.

## 4.3 Signal Integrity

In order to facilitate seamless system scaling to the higher-speed 10 Gbps MGTs included in the Virtex-4 and later devices, as discussed in Section 2.6, it is necessary to design a carrier board capable of transmitting and receiving 10 Gbps serial data. Without the aid of fiber optics, this is a difficult task, and with only FR-4 and copper it becomes a monumental problem.

In traditional supercomputer design, all carrier board I/O is routed to the chassis backplane, where cable assemblies are attached to interface with the system. However, in order to avoid using more expensive PCB dielectrics than FR-4, or being forced to use extremely expensive fiber interconnects we route I/O to both edges of the carrier board, which minimizes high-speed transmission line lengths to fewer than 6 in. long, and we use high quality connectors and cable assemblies along with active equalization to achieve an extremely low BER at a low component cost. The disadvantage of this is that the bottom edge mezzanine cards on the carrier board cannot have their system I/Os routed to the top edge of the board. The only cost of this decision is increased time to deploy and maintain the system, as replacing a board requires unplugging cable assemblies. Furthermore, if we did route these signals from the bottom edge to the top edge of the carrier

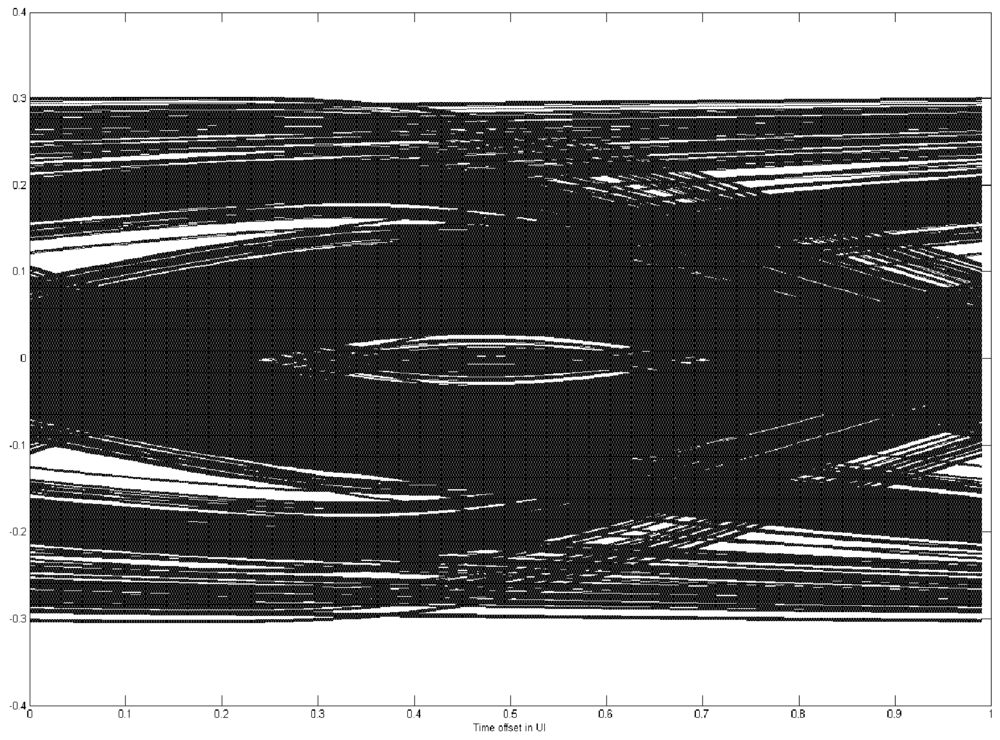

**Figure 4.3: 10 Gbps Eye Diagram for the Bottom-Edge Carrier Board I/O Routed to the Top-Edge**

board, the BER would be atrocious at 10 Gbps as seen by the lack of any opening in the corresponding eye diagram displayed in Figure 4.3. To engineer around this, we would need active equalization circuitry on the midplane prior to the long cable run in addition to carrier board active equalization at the receiver. However, using two equalization stages is a poor design decision, as the channel BER would increase manyfold and the channel would become nearly impossible to accurately simulate.

In order to minimize transmission line impedance variation and signal crosstalk, the carrier board 10-layer stackup is conservative, where all routing layers are separated from each other by a reference plane as shown in Figure 4.4. Additionally, trace lengths are always minimized, and the longest high-speed signal trace is fewer than 6 in. long. However, to compensate for both dielectric and skin effect losses over long cable runs, active equalization circuitry is employed on the carrier board via the Maxim 3804 12.5 Gbps receiver equalizer [38]. Using this active equalizer, both the 2.5 Gbps and the 10 Gbps eyes are wide open with a 30 ft high-quality cable run. The eye diagrams for these channels are shown in Figures 4.5 and 4.6.

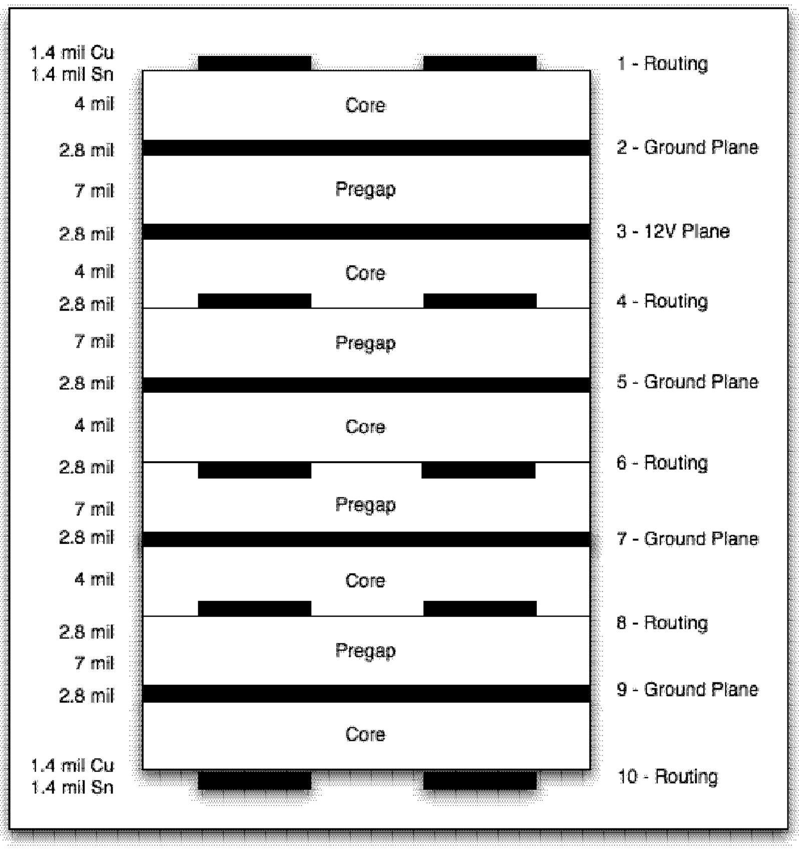

**Figure 4.4: Carrier Board Layer Stackup**

These simulations accurately model all aspects of the channels from the driver to the package leads, vias, connectors, transmission lines, cable assemblies, and receiver.

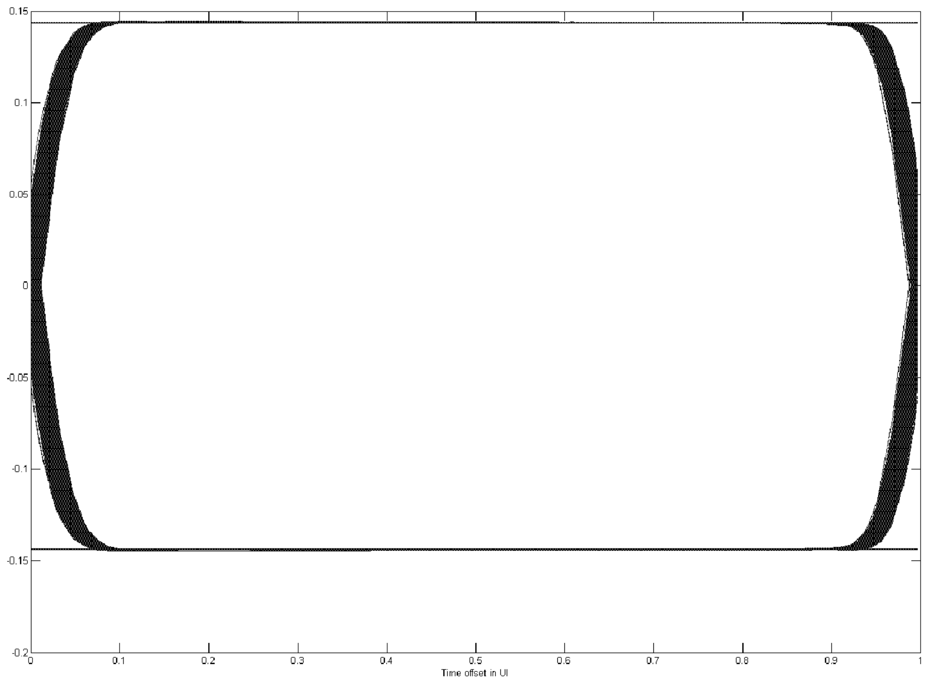

**Figure 4.5: 2.5 Gbps Full Channel Simulation Over 30 ft Cabling**

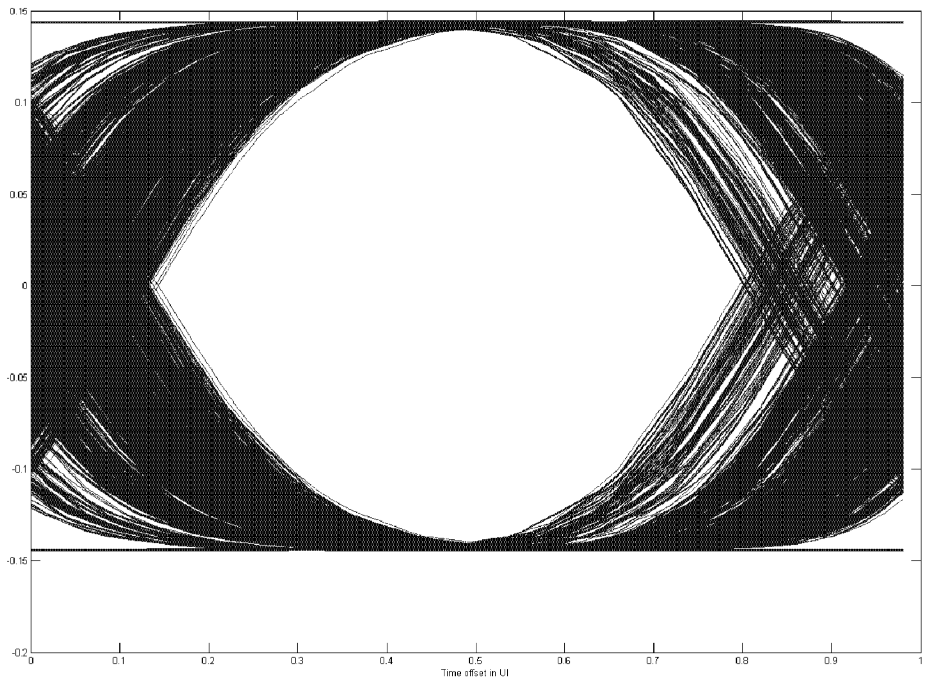

**Figure 4.6: 10 Gbps Full Channel Simulation Over 30 ft Cabling**

# Chapter 5

# Chassis, Subsystem, and System Design

Many of the critical design decisions were made at the mezzanine card and carrier board level in order to facilitate scaling to larger systems. Very few systems other than the ILLIAC 6 platform are able to successfully scale in size by several orders of magnitude. Moreover, it is extremely uncommon to provide such a high degree of scalability using the same electrical and logic protocols for communication at every level in the hierarchy, and providing the necessary drive strength to do so at the smallest module in the system is simply unheard of. Systems generally incorporate highly complicated communications hardware to perform signal buffering or adaptation at each level in the hierarchy, which potentially reduces bisectional bandwidth and adds a great deal of complexity to the overall design. However, our novel architecture and physical design make seamless scaling a reality. In this chapter we overview the design of the chassis and we discuss the important aspects of larger system scaling including system interconnects, power distribution, thermal relief, and signal integrity.

## 5.1 Chassis Overview

Much like the carrier board, the chassis can be configured as a full computer with I/O on both sides, as an ingress or egress point, or as a component in a larger system. As a full system, a populated chassis is an extremely powerful computer. It contains 256 FPGAs, 1024 DSPs, and 128 GB of RAM, and it dissipates approximately 8 KW.

A populated chassis consists of 32 carrier boards and four power boards residing in two 23 in. telco enclosures bonded together and connected with a custom midplane with 16 blades per enclosure. The front enclosure is oriented so that blades slide in parallel to the floor. However, the rear enclosure is rotated 90° along the axis orthogonal to the midplane with blades sliding in orthogonally to the floor. Figure 5.1 depicts the chassis organization. The midplane implements an enormous 256 x 256 full-shuffle exchange network, and rotating the rear enclosure by 90°results in a geometry that greatly simplifies the midplane design as each connection through the midplane is simply a via.

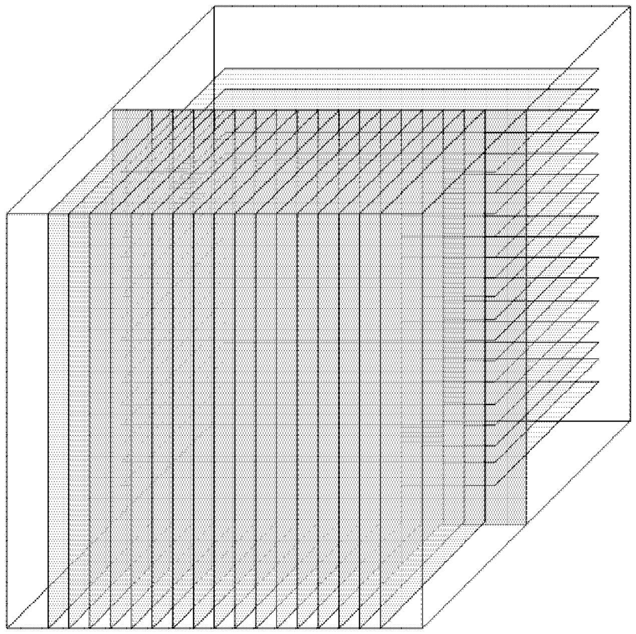

**Figure 5.1: Populated Chassis Organization**

The midplane contains 16 rows of VHDM connectors and Molex connectors on the front side and 16 columns of VHDM and Molex connectors on the back side for attachment of the 32 carrier boards, and the midplane stackup includes four ground and four 12 V planes for power distribution. The midplane also includes a 1 Gbps ethernet jack and an ethernet switching network for connecting to the 256 modules via their Ethernet programming and debugging interface. Lastly, the midplane contains two -48 VDC connectors for powering the entire chassis.

## 5.2 Power

Each chassis is powered with a high-current capacity -48 VDC source. A small power management and distribution circuit on the midplane controls power distribution to all connected boards. This manager negotiates with each connected carrier board and provides a number of services including over-current protection, hot-swap support, and soft-start power ramping to avoid surges. Ultimately, proper design of the power distribution system will require detailed simulation.

Powering a subsystem requires at minimum a 64 KW feed, and a full system requires over 512 KW. Power demands of this nature require a dedicated power substation and high-voltage AC power lines. We have not yet explored the design of the power system for an ILLIAC 6 computer of this size, but most supercomputer centers are designed to support powering systems as large as an ILLIAC 6 subsystem or system. Furthermore, a number of systems with similar or larger power requirements have been designed and implemented over the last 30 years, so providing power to a full ILLIAC 6 is certainly feasible.

## 5.3 Thermal Relief

The module design incorporates thermal relief via heat sinks and conduction into the ground planes as discussed in Section 3.1, but it is the responsibility of the chassis to provide the necessary air flow to sufficiently cool the modules. Our chassis consists of two ATCA shelves, and their power budgets falls within the ATCA design specifications, so the thermal budget should as well [6]. The air cooling guidelines and thermal models defined in the ATCA specification can be used to design the chassis cooling system. We have not explored the details of how to provide thermal relief for a subsystem or a system, but as with powering a subsystem or system, existing systems with similar thermal constraints to the ILLIAC 6 provide proof of feasibility and design guidelines for cooling the ILLIAC 6.

## 5.4 Signaling

As with the module and carrier board, ensuring signal integrity at the chassis level is critically important. The only critical channels in the chassis are the high-speed MGT connections through the midplane. We have modeled these channels from FPGA egress on the upper row of modules on a front carrier board to FPGA ingress on the upper row of modules on a rear carrier board. As discussed in Section 4.3, these channels must be kept short in order to avoid active equalization, and the geometry of the chassis and midplane make this possible. The eye diagrams for both 2.5 Gbps I/O and 10 Gbps I/O over these midplane channels are shown in Figures 5.2 and 5.3, and both eyes are wide open.

One of the primary difficulties of interconnecting large computer systems with high-bandwidth connections is the cost of the connection medium. The price, size, weight, bandwidth, and reliability of different mediums vary drastically from low-cost unshielded copper to high-cost glass multimode fibers. In the context of interconnecting chassises, it is clear that the cable density of 512 unidirectional high-speed channels is a critical design constraint. With the current module revision, these channels operate at 2.5 Gbps, but these channels must be able to support the 10 Gbps MGT I/O incorporated into future modules.

Many computers utilize fiber optics for signaling between systems similar in size to an ILLIAC 6 system or even subsystem, but the component cost for fiber transceivers and cables is currently orders of magnitude greater than the component cost for a copper solution. Furthermore, the packaging density of high-speed

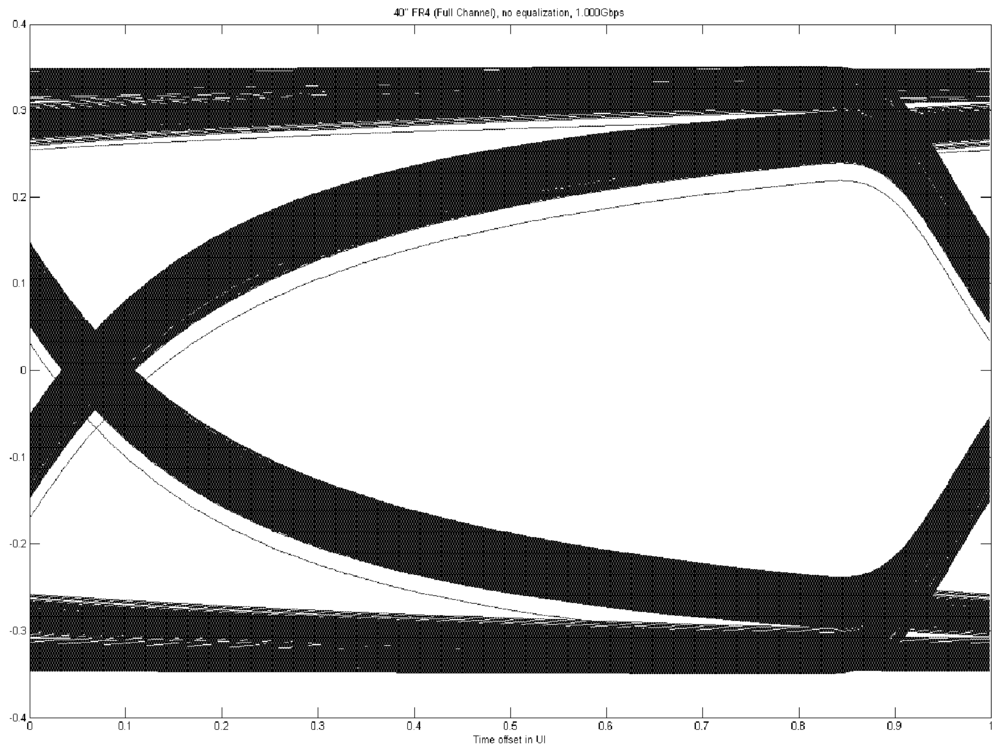

**Figure 5.2: 2.5 Gbps Eye Diagram for the Midplane Channel**

copper cable connectors and assemblies is much greater than for comparable optical solutions. Moreover, the highest speed fiber modules and cables applicable to this system can support 10 Gbps per fiber, so a full chassis to chassis connection in the context of a system would require 512 fiber-optic cables.

A common copper interconnect solution would be to use 10 Gbps Infiniband, but 10 Gbps Infiniband is actually an aggregate bandwidth of 4 seperate 2.5 Gbps channels, so this solution would require approximately four times the conductors of a truly serial 10 Gbps protocol. Furthermore it would either quadruple the number of MGTs and quarter their effective bandwidth, or it would require special hardware to split each 10 Gbps MGT channel into four 2.5 Gbps channels at the sender and hardware to merge these channels back together at the receiver. The same problem is true of 10 Gbps Ethernet. In terms of signaling conductor count, both of these solutions use differential signaling and require a single pair of copper conductors for each channel resulting in at least 4,096 copper conductors to connect two chassises together in a system. If each of these conductors is a 24 AWG wire, and ground connections and shielding are taken into account,

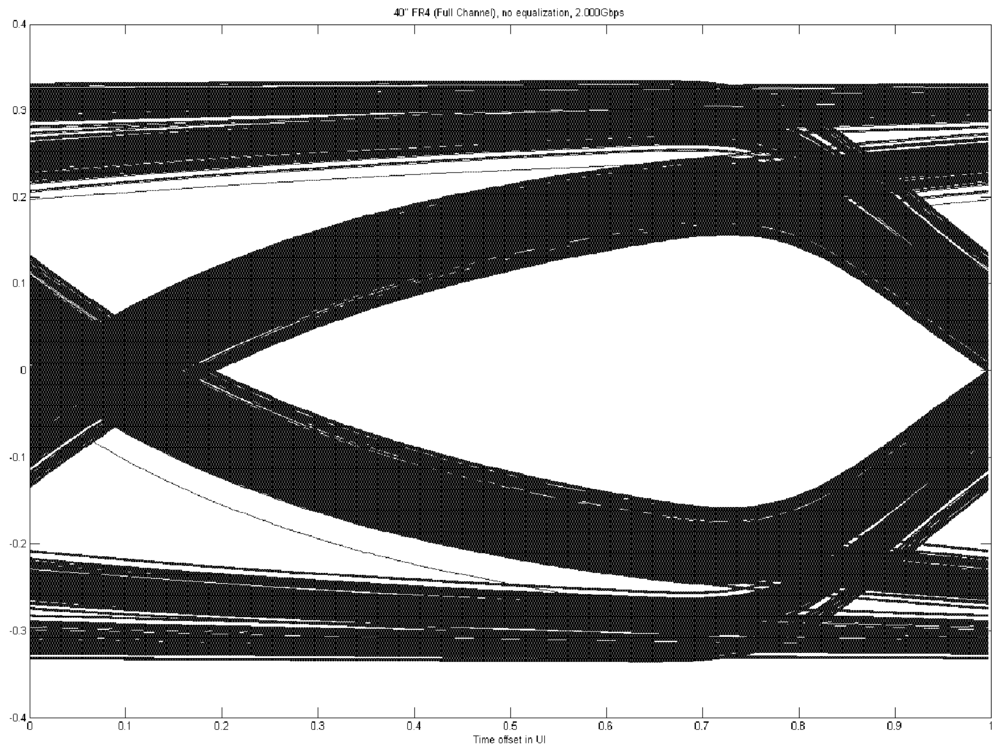

**Figure 5.3: 10 Gbps Eye Diagram for the Midplane Channel**

these 4,096 wires become an enormous physical cost in terms of size and weight. It may not be possible to reliably connect such a system.

The solution we use in the ILLIAC 6 platform is to simply signal at 10 Gbps over each differential pair, restoring the signal using active equalization and amplification circuitry at the receiver. This solution requires only 1024 signaling conductors for a chassis to chassis connection in a system, which is non-trivial, but certainly feasible. Therefore, by working through the electrical difficulties of using high data-rate copper, we have designed a system that has uncompromising bandwidth scaling similar to fiber, but at the greatly reduced cost of copper. The soundness of these channels is discussed in Section 4.3.

# Chapter 6

# Communication Architecture

The network interface is one of the most important aspects of modern computing platforms. Systems often utilize a great deal of hardware to provide a robust high-bandwidth and low-latency network infrastructure. For example, the IBM Blue Gene/L [26] is normally configured with half of its processors dedicated as communication interfaces. In the ILLIAC 6, we dedicate only approximately half of the FPGA resources and no processor resources to this task, which is an enormous savings compared to a system like the Blue Gene/L. In this chapter, we overview the ILLIAC 6 communication architecture and detail its core implementation in the FPGA fabric.

## 6.1 Communication Architecture Overview

The communication fabric of the ILLIAC 6 is unique in that it provides an extremely low effective bit error rate and seamless integration of reconfigurable computations in a simple, high-performance design. There is a single unified architecture for the entire platform, but it is useful to logically separate the system core communication from the system I/O. All core communications to and from a module pass through the module's central FPGA via high-speed MGTs. All I/O communications to and from a module pass through either the high-speed MGTs or the 50 Gbps dedicated I/O port. For core communication the MGTs are tied, inside the FPGA, to communication channels which perform error detection and retransmission for extremely low effective bit error rates. For system I/O the MGTs and/or the 50 Gbps I/O port can be mixed and matched to form 8 logical I/O channels leaving a module and 8 logical I/O channels entering a module. The system core and system I/O channels form two 8x8 crossbar switches, one for the up-network and one for the down-network. Each crossbar switch connects four ingress MGTs or logical I/Os and four ingress TigerSHARC link-ports to four egress MGTs or logical I/Os and four egress TigerSHARC link-ports.

The FPGA plays a central role in the communication architecture, because at the minimum it is responsible for the routing and switching of frames. However, by designing a simple modular architecture for the FGPA communication infrastructure, we are able to directly support a plethora of I/O devices, and we

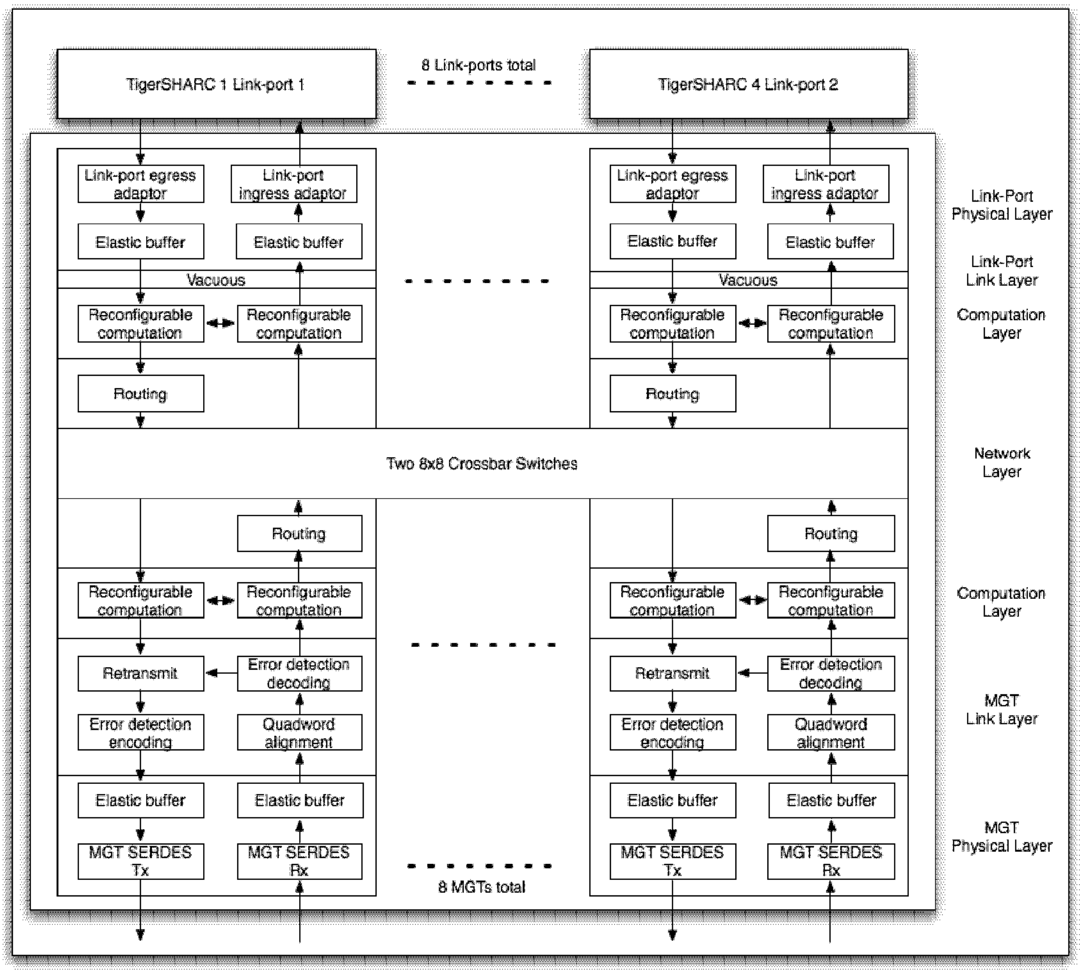

**Figure 6.1: FPGA Architecture for Core System**

are able to expose reconfigurable fabric to the programmer as both a communication and a computation resource. We have organized the functionality of the FGPA as a *network stack* as detailed in Figure 6.1. There are several compelling reasons to organize the communication fabric of the machine in this way:

- It allows the design to be easily broken into pipelined modules.
- It allows us to account for the different characteristics of the various attachments to the module (from the processors, from other modules, from I/O devices).
- It allows us to integrate reconfigurable computation blocks into the communication fabric while maintaining the integrity and performance of the communications.

## 6.2 Physical Layer

Any I/O entering or exiting the FPGA passes through a physical network layer. The physical layer converts all I/O protocols to the word-oriented parallel protocol with a flow-control mechanism that is used within the FPGA. The physical layer for 50 Gbps system I/O, MGT I/O and for TigerSHARC link-port I/O are very different from one another, but data flows into and out of their respective link layers with the same width and protocol. A module configured as a system core device is shown in Figure 6.1. It is possible to define a physical layer for nearly any system I/O device. A module configured as a system egress or ingress device with 10 Gbps Ethernet is detailed in Figure 6.2. The 50 Gbps I/O connection supports digitally controlled line termination and can be configured to signal via nearly any 1.5V, 2.5V, or 3.3V electrical protocol, and the physical layer can be programmed to speak nearly any logical protocol. Furthermore, for system I/O the physical layer is only constrained internally to 16 unidirectional or 8 bidirectional logical channels. Multiple I/O devices can easily be aggregated into a single channel by the physical layer. A module configured as both a system ingress and egress device with aggregated USB I/O and 10 Gbps Ethernet is shown in Figure 6.3.

## 6.3 Link Layer

The system I/O link layers can vary greatly depending on the device being interfaced with, and the logical separation between the link layer and the physical layer may not be clear. However, in our network stack, the link layer is always a separate module and is generally responsible for reliable communication and flow control.

The core system link layers include the MGT link layer and the TigerSHARC link layer. The TigerSHARC link layer is vacuous, as careful design and simulation ensures that the effective BER of these channels is negligible. The MGT link layer is more complicated, because the MGT physical layer and the underlying high-speed channel are inherently lossy. These channels utilize ultra-high-speed serial communication between FPGAs over FR-4 PCBs, connectors, and cables, so even with active equalization, under the right conditions, the BER can fall below $10^{-10}$. This rate necessitates robust error correction and detection that can only be achieved through a reliable transmission protocol. We can expect the multiple bit-flips within a word, and total loss of parts of words in the worst case, so both robust error detection codes and a retransmission mechanism must be employed. In order to keep the overhead of this mechanism low, we add 4 bits of error detection per word and utilize a simple sliding window-based retransmission protocol.

At the sender, the minimal unit of data entering the link layer from the network layer is a quad-word

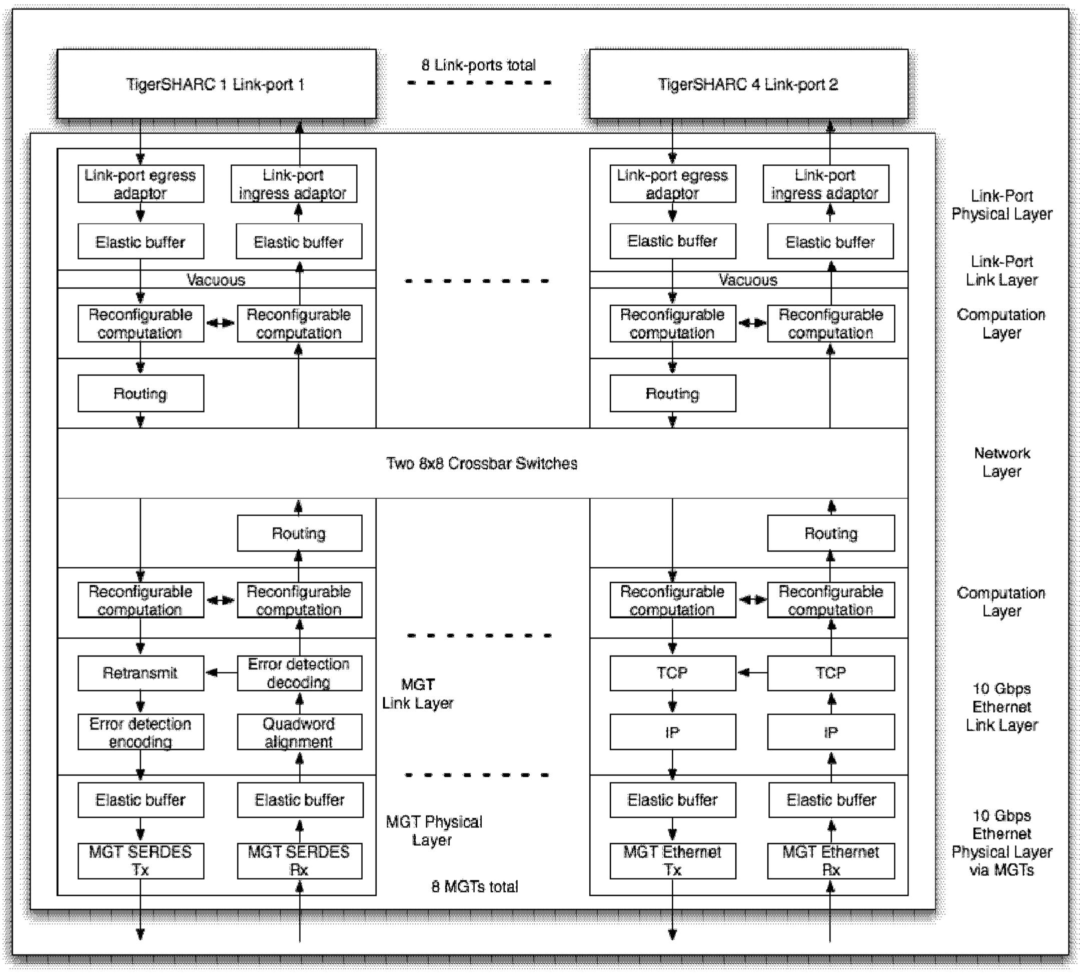

**Figure 6.2: FPGA Architecture for Core and System I/O**

(128-bits). It is the responsibility of the link layer to guarantee error free delivery of this quad-word to the network layer of the receiver. To realize this, the link layer appends an additional word of data to this quad-word fixing the frame size at 5 words. This extra word includes 16-bits of error detection, 14-bits of sequence numbers, and 2-bits of identification: IDIN and IDOUT. Figure 6.4 shows the format of this word.

In order to implement the sliding window, a 7-bit sequence number is utilized. Each frame carries its own sequence number: SEQIN and the sequence number of the last frame successfully received: SEQOUT. The choice of a 7-bit sequence number does not imply the need for frame buffers with a depth of exactly $2^7$ frames. Rather, it is an upper bound on the frame buffer size. The minimum frame buffer size is determined by the latency of the underlying communication channel. The frame buffers must be large enough, so that when data must be retransmitted, it is still available in the buffer. A more natural 8-bit sequence number

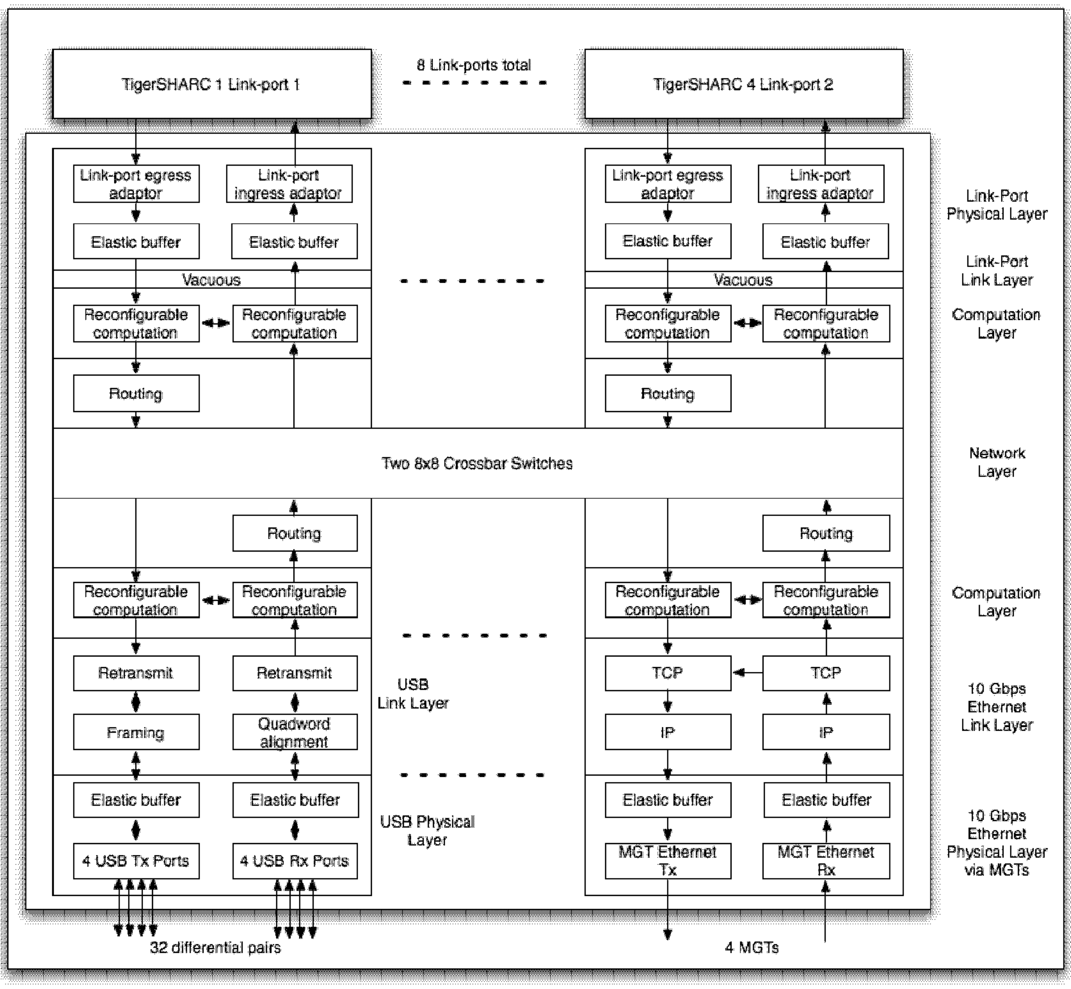

**Figure 6.3: FPGA Architecture for System I/O**

would be convenient, but this would necessitate expanding the header beyond a single word, which is quite problematic.

Nodes must acknowledge every message with either an ACK, indicating successful reception of a frame, or a NAK indicating unsuccessful reception or a buffer-full condition. An ACK (0101010) or a NAK (1010101) is transmitted by performing an XOR of the ACK/NAK with SEQOUT. A receiver will always know the expected SEQOUT and can recover the ACK/NAK by performing an XOR of the received SEQOUT/ACK field with the expected SEQOUT. Not only does this conserve bits, but it also adds an extra layer of protection over the SEQOUT and ACK/NAK.

The 1-bit IDIN and IDOUT fields indicate whether the frame contains an ACK/NAK and whether the frame contains valid data. It is possible to transmit a frame of data that does not include an ACK/NAK. It

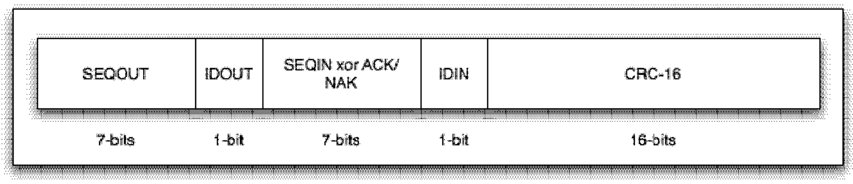

**Figure 6.4: Reliability Word**

| IDIN | Logical Meaning |
|---|---|
| 0 | Invalid Data |
| 1 | Valid Data |

**Table 6.1: IDIN Truth Table**

is also possible to transmit a frame that consists solely of an ACK/NAK and contains no valid data. Refer to Tables 6.1 and 6.2 for details.

To achieve a high degree of error detection a 16-bit cyclical redundancy check, CRC-16, is utilized. CRC-16 is capable of detecting all single and double bit errors, all errors with an odd number of bits, and all burst errors of length less then 16. The CRC-16 is performed over the entire message including the ID, SEQIN, and SEQOUT.

All connected nodes in the system are connected by two opposing independent unidirectional channels. However, in order to implement a reliable transmission protocol, it is necessary to bond these channels in the link-layer into a single high-speed bi-directional full-duplex channel with a bandwidth equal to the sum of the unidirectional component channels.

Both ends of the channel are frame-synchronized by sequence numbers which are initialized to zero. After every frame is transmitted the sender's sequence number is incremented modulo 128. Upon reception of a frame, the receiver checks the received sequence number against the expected sequence number, and if the numbers match, the message is acknowledged, and the expected sequence number is incremented modulo 128.

These sequence numbers are necessary in the implementation of a sliding window protocol. A simple stop-and-wait protocol underutilizes the channel bandwidth and increases latency. In a sliding window protocol, a finite window of frames can be transmitted before an acknowledgment is received as shown in

| IDOUT | Logical Meaning |
|---|---|
| 0 | Invalid ACK/NAK |
| 1 | Valid ACK/NAK |

**Table 6.2: IDOUT Truth Table**

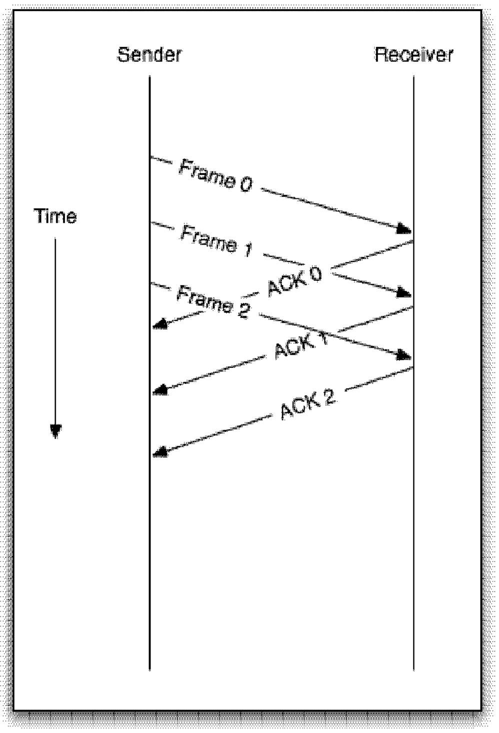

**Figure 6.5: Sliding Window Timeline**

Figure 6.5; this is somewhat analogous to the pipelining in a processor.

Lastly, the sequence number and reliability word provide a robust and natural end-of-frame boundary, and could be utilized to provide a means of frame synchronization. However we have offloaded the task of providing frame synchronization to the physical layer in order to simplify the link layer architecture.

As each quad-word enters the link-layer from the network-layer, a single word of reliability data is appended, resulting in a 5-word frame. This frame is pushed onto the head of a hardware frame buffer. The size of this frame buffer defines the length of the logical sliding window. The frame remains in this buffer until an ACK is received, at which point it is removed from the buffer.

Furthermore, since the underlying channel is lossy, both messages and acknowledgements can be lost in transmission. In order to account for this, each frame in the frame buffer has an associated timer. When a frame is added to the frame buffer, the timer begins to count down. If no ACK/NAK is received before the timer times out, the timed-out frame and all successor frames in the window are retransmitted. Additionally, a retry count is maintained for each frame. If the count reaches a specified threshold, a critical error is signaled.

In the receiver, data is copied from the physical layer to the link-layer one word at a time into a small incoming frame buffer. A large buffer is not necessary, because out of order frame delivery is not permitted; this means that the receiver does not need to maintain an incoming sliding window. When an entire frame is present in the frame buffer, the CRC-16 field is used to verify the integrity of the message. If the message is corrupted, a NAK, including the expected SEQIN is transmitted back to the sender.

Every frame must be acknowledged, and the reliability word has a field just for this purpose. The ACK/NAK for an incoming frame can simply be included in the next outgoing message. If there are no outgoing message in the egress frame buffer, then a dummy message is created and the reliability word is appended. This dummy message has a negligible effect on channel throughput as it is only transmitted when the send buffer is empty. The justification for this is that transmitting the reliability data by itself requires that the receiver be capable of receiving both 5-word and 1-word messages. This appears straightforward since data can flow into the link-layer from the physical layer one word at a time, and on reception of a new message, the hardware can always attempt to decode the first word as a reliability word. The downside of this is that there is a $2^{-32}$ probability that the first word in a 5-word message is exactly the same bit sequence of an acknowledgment of the last sequence number. This would ultimately result in a lost frame if the message should have been NAK'd but is ACK'd.

The constraints for the dummy frame are that the first word must be all zeros, and the ID bits must be set appropriately to indicate invalid data. Furthermore, if a frame is to be transmitted, but no ACK/NAK is ready, the message is transmitted with the ID bits set to indicate no ACK/NAK is included. Additionally, SEQIN is populated with a value known to be invalid: (expected SEQIN $\oplus$ 1111111).

## 6.4 Computation Layer

Integrating reconfigurable computation into the network stack provides a very clean architecture and programming model. Any computational unit that can operate at the bandwidth of the communication channels can be bolted into the channel at, in effect, no cost, as the latency of communication is increased only slightly and the bandwidth is unaffected. Computational blocks (components) can be created and associated with the each ingress and each egress port of the FPGA as discussed in Section 8. Components that have multiple output streams can be mapped to the ingress ports of the FPGA, where their outputs will feed the 8x8 crossbar that resides behind the ingress port. Components that have multiple input streams can conversely be mapped to the egress ports of the FPGA, where their inputs are fed by the 8x8 crossbar that precedes the egress port. The natural width of each computation unit is a single word, but units can be arbitrarily pipelined and perform operations on nearly any data width. Lastly, computational blocks can be easily

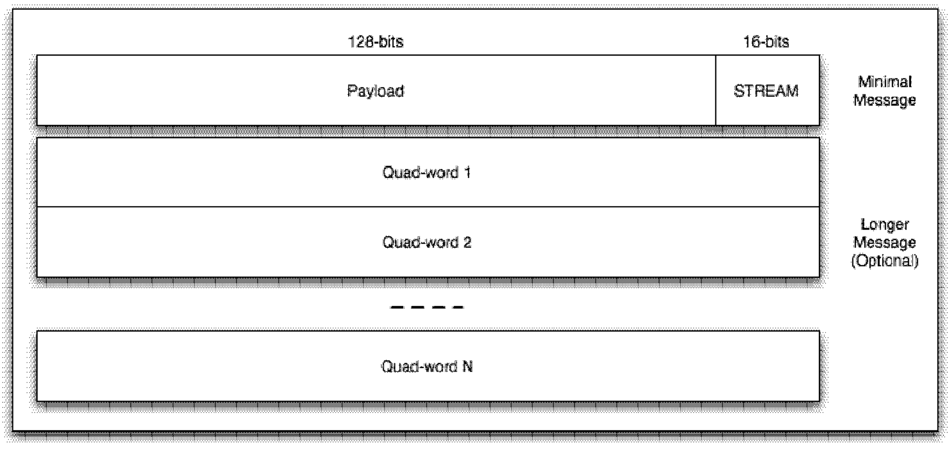

**Figure 6.6: Message Format**

bypassed at the message level and can implement limited data-dependant control in order to allow their functionality to be conditionalized by data type, etc.

Approximately 50% of the Virtex-2 Pro fabric is available for the reconfigurable computation layer, which is more than enough silicon to perform a great deal of realtime computation. Furthermore, the novel location within the network stack provides for a plethora of new and unprecedented communication and computation possibilities. It is easy to see the power of this layer in terms of adaptive and modifiable networks, because it is trivial to perform operations such as stream merging and forking, and it is not difficult to utilize this layer to change the network topology and even generate virtual channels.

The computation layer also plays a critical role in system I/O. The network layer discussed in Section 6.5 schedules and routes streams. Modules functioning as system I/O points utilize the computation layer to frame and route these streams before or after entering the network layer. This is an essential part of maintaining the separation of concerns for each layer, because it frees the physical and link layers from this responsibility, which facilitates the use of existing IP cores for system I/O.

## 6.5 Network Layer

The network layer performs routing of messages over the crossbar switch. A number of different routing protocols can be implemented in this layer, but the most practical for real-time applications is static routing with fixed message priorities. The smallest logical message that can be transmitted in the core system as a 128-bit quad-word as depicted in Figure 6.6. 16-bits of this message are a stream referred to as STREAM.

$$D = \frac{S*T}{P} + F$$

**Figure 6.7: Routing Policy Deadline**
**A: Arrival time of the message in clock cycles**
**P: Message priority**
**S: Message size in bytes**
**T: Clock cycles required to transfer a byte through the channel**

This identifier is unique for all communication streams that can share a link, but it may be reused in messages that have no channels in common. For a given channel, the FPGA determines both the message length and the destination node by using table lookups on STREAM. Longer messages simply follow the minimal message with the number of quadwords indicated by STREAM.

Messages must arbitrate for outbound channels, so a priority mechanism must be employed to ensure fairness and forward progress. The simplest solution for real-time applications is to use a fixed priority determined at compile time and programmed into the routing tables at application load time. The priority, $P$, is defined as the inverse of the percentage of bandwidth required for this STREAM. When a message arbitrates for an output port, it can be assigned a deadline $D$ as defined in Figure 6.7. The scheduler operates using each clock cycle as a time tick. In this way, the scheduler does not need to be altered if the FPGA fabric clock is slowed to reduce power consumption.

An EDF scheduling policy will minimize the lateness of all of the messages even without preemption [32], so if the bandwidth requirements for all message types that exit a port do not exceed the bandwidth of the port, EDF provides an optimal fair solution to the arbitration problem. This policy is, in effect, a contract between the program mapper (see Section 8) and the switches, as it is the program mapper that allocates channel bandwidth based on application performance requirements. The switches must implement the program mapper's allocation. To the extent that the bandwidth of a communication channel is underutilized, the clock rate on that channel can be reduced. To this end, our design maintains separate clock domains inside the FPGA (from the link layer and above) so that a clock manager can reduce the clock rate of the FPGA without affecting application bandwidth guarantees. This, too, is a contract between the hardware and the program mapper, which is responsible for setting the clock rate of the communication channels when an application is loaded.

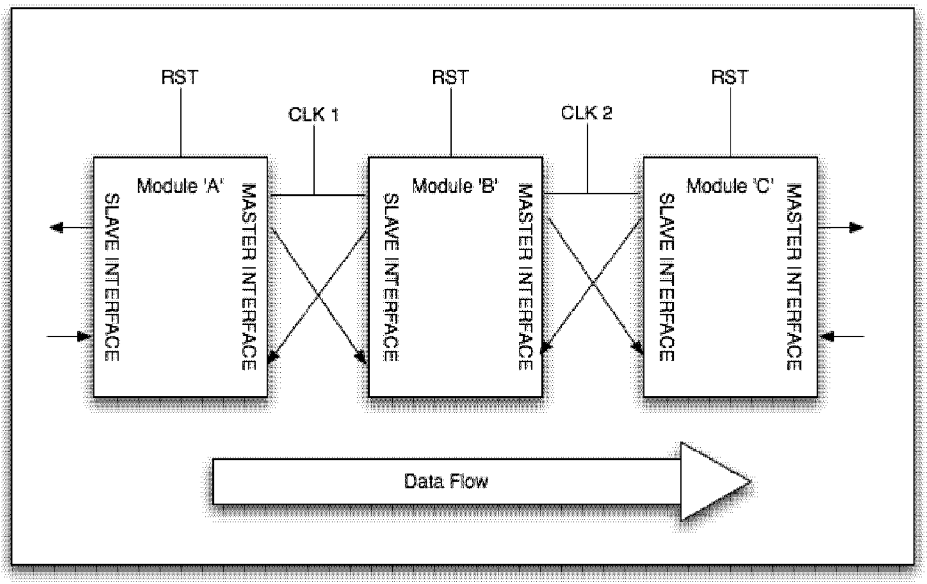

**Figure 6.8: Module Dataflow Architecture**

## 6.6 Realization of the Architecture

The network stack based FPGA architecture is easy to implement using a combination of existing IP cores and custom cores. Other than the external interfaces of the physical layer, all of the layers detailed in Figure 6.1 are connected via a single word wide bus and use handshaking for flow control. Each module in each layer of the stack corresponds to an IP core in the FPGA, and each module can be arbitrarily pipelined. Most preexisting IP can be be directly utilized with simple protocol adapting wrappers. Reconfigurable blocks (components) that are created by the programmer are automatically wrapped in such a protocol adapation wrapper.

### 6.6.1 Module Interfaces

All modules are interconnected with the standardized buffer interface shown in Figures 6.8 and 6.9. With this architecture, the egress port of a module has a MASTER interface, and the ingress port has a SLAVE interface. Data is passed synchronously via a shared clock between each MASTER and SLAVE. Time-domain crossing modules are much more difficult to design and verify than our standard modules, so only special time domain crossing modules like the fully asynchronous buffer, shown in Figure 6.11, can be used to cross clock domains. The rules governing the minimal module interface are:

- I/O signals must end with '_I' or _O to indicate inputs and outputs.

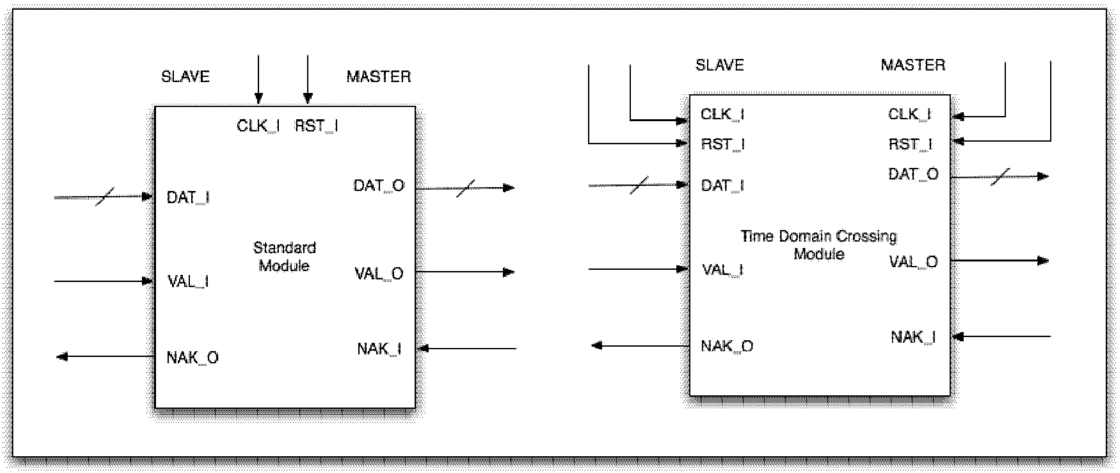

**Figure 6.9: Module Interface Details**

- Multi-byte channels must be little endian (the most significant byte is in the highest address).
- Only positive-edge triggered flip-flops can be used to hold state.
- Modules must transfer data synchronously on the positive edges of the clock [CLK].
- Modules must support a synchronous reset [RST], which causes the module to return to its initial state.
- Modules must have a data interface [DAT] with a width of 32-bits.
- Modules must have a data valid signal [VAL]. When a master asserts [VAL_O], a valid data transfer is in progress; it must remain asserted until the completion of the transfer.
- Modules must have a flow control not-acknowledge signal [NAK]. When [NAK_O] is asserted by a SLAVE, the SLAVE must be able to accept data on the current cycle but will not be able to accept data on the next cycle.

### 6.6.2 Synchronization

Moving data between two asynchronous time domains is never a trivial task; it can be reduced to the same problem as sampling data with a flip-flop. If the data being sampled is in transition, the flip-flop can enter a metastable state, with the output of the flip-flop being in an intermediate state until noise sources induce enough of a perturbation voltage to move the flip-flop out of metastability. If the output of the flip-flop is

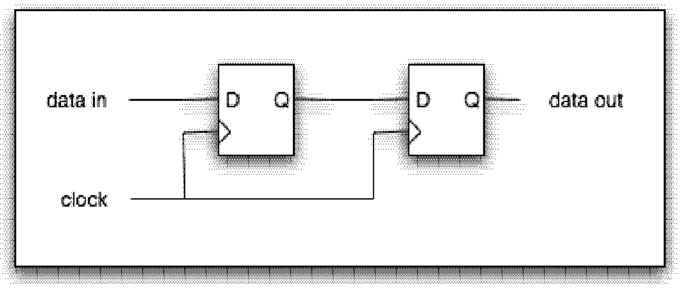

**Figure 6.10: Synchronization Primitive**

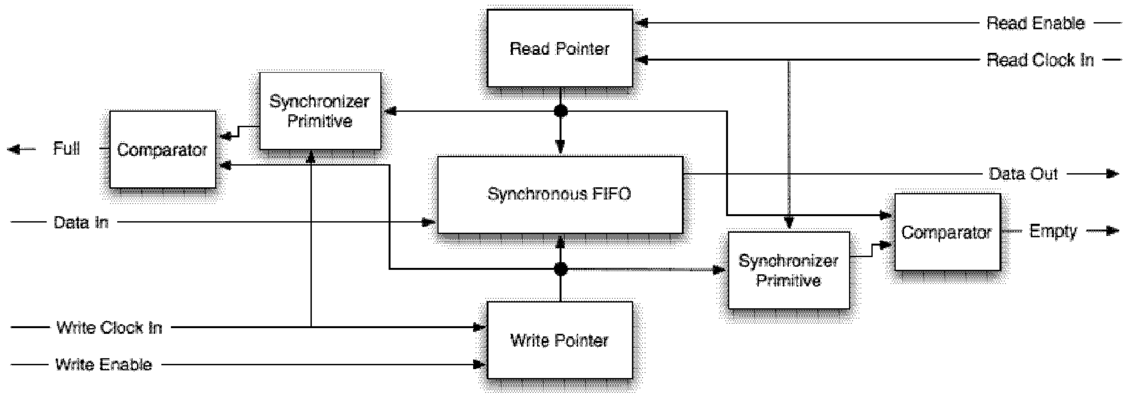

**Figure 6.11: Custom Asynchronous FIFO**

sampled before settling to a valid logic value, the digital circuit will fail. Moving data across asynchronous time domains reduces to sampling data with an unknown data to clock phase relationship, and hence results in a finite probability of sampling the data during transition, which results in metastability and perhaps synchronization failure [55].

Data that crosses between the physical layer and link layer must cross the clock domains between the I/O device and the remainder of the FPGA fabric. The physical layer is always responsible for synchronization - providing a safe means for data to cross clock domains. This is easily accomplished for I/O clocked at 300 MHz or slower with a custom fully asynchronous FIFO buffer inside the Virtex-2 Pro. With careful design it is straightforward to ensure that this elastic buffer has an acceptable synchronization failure rate, i.e., much lower than the expected failure rate of the processors or memory of the system.

Our asynchronous FIFO design, shown in Figure 6.11, utilizes a standard two flip-flop synchronization primitive as depicted in figure 6.10. The MTBF of this primitive is given by Figure 6.12 [55]. The flip-flop

$$T = \frac{e^{\frac{t_w}{\tau}}}{t_a f_a f_b}$$

**Figure 6.12: Synchronization Primitive MTBF in seconds**
$t_a$: **Flip-flop aperature time**
$f_b$: **Clock frequency**
$f_a$: **Data frequency**
$t_w$: **Minimal wait time before sampling synchronized data**
$\tau$: **Regenerative time constant of the flip-flop**

aperture time, $t_a$, and the regenerative time constant, $\tau$, are fixed for each flip-flop type within an FPGA, and there are only two types of flip-flops found within the Virtex-2 Pro: IOB and CLB flip-flops. Xilinx does not release the internal details of these flip-flops but has published some results of the observed MTBF in the form of a graph for them [42]. By extrapolating the results of [42] to Figure 6.12, it is possible to estimate the MTBF for any synchronizer in the Vitex-2 Pro, including our synchronization primitive as depicted in Figure 6.13. The custom Asynchronous FIFO uses CLB flip-flops and performs only a single synchronization operation for each read or write to the buffer, so its synchronization MTBF is identical to the synchronization primitive. It is clear from Figure 6.13 that the MTBF for this FIFO operating at 300 MHz is approximately $10^{20}$ s. With a full ILLIAC 6 system, at most $10^6$ Asynchronous FIFOs are required, resulting in an expected synchronization MTBF of $10^{14}$ s, which is many times longer than the expected life of the system.

With clever design it is possible to parallelize synchronous I/O before crossing clock domains, in order to reduce the required synchronization frequency. This is a viable option, but it requires a custom solution for each interface, more than twice as much fabric, and increased design time for floor planning and layout. Since the TigerSHARC interface operates at 500 MHz, this solution would be necessary to achieve a reasonable MTBF. However, in order to save resources and design time, we chose to avoid synchronization for this interface and move data synchronously between the TigherSHARCs and the FPGA phsyical layer by distributing matched low-skew clocks to the DSPs and the FPGA as detailed in Section 3.2.

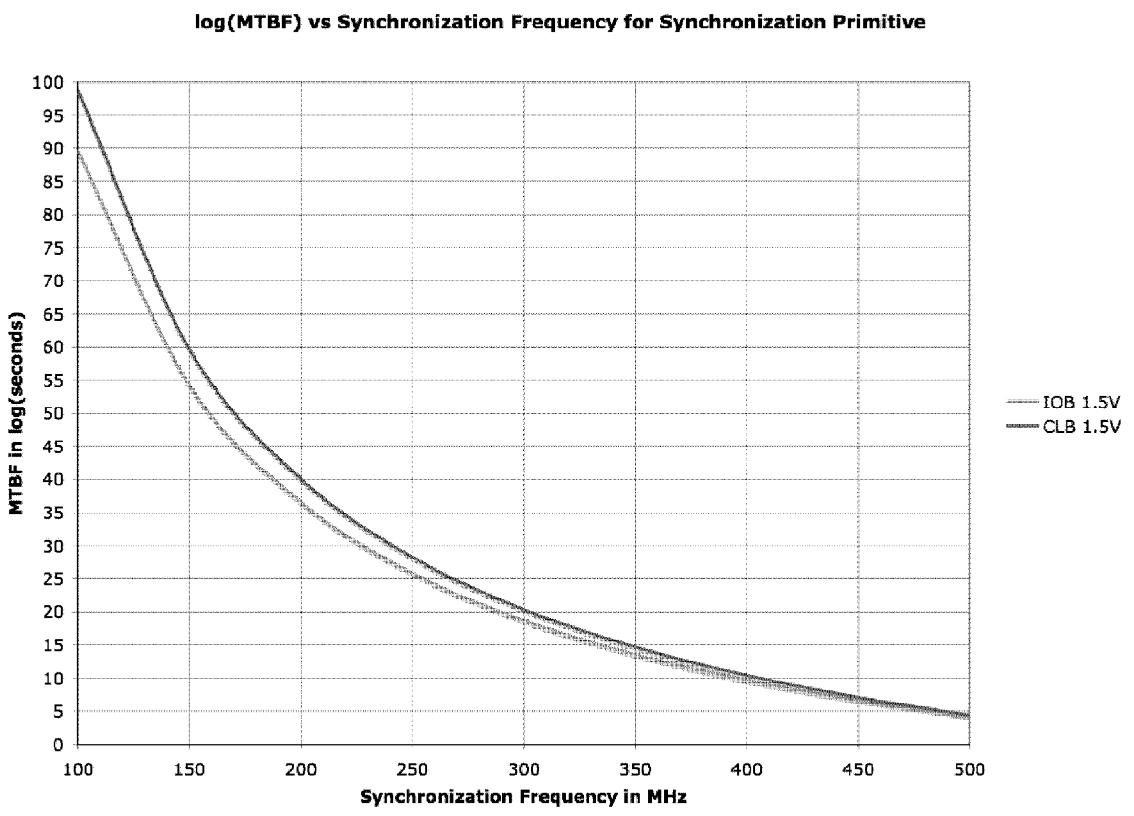

Figure 6.13: Synchronization Primitive MTBF

# Chapter 7

# Machine Maintenance

Like most supercomputers, the ILLIAC 6 requires a host machine to boot, program, monitor, and debug the system, so every module includes a 10 Base-T Ethernet interface for connecting to the host. As depicted in Figure 2.1, this ethernet interface is connected to the MCU via a single chip hardware TCP/IP stack. Having this low-power, low-speed ATMega1280/V MCU on every module provides a robust means of system control. Since it operates at 8 MHz, it consumes little power and it is the most reliable active component on a module. It can detect and correctly handle brown-out conditions, and global watchdog timers can be used to detect and correct critical errors resulting in a lack of forward progress. As detailed in Figure 7.1, the MCU controls the DSP and FPGA reset pins, and it acts as a JTAG emulator for the FPGA and DSP JTAG chain. The MCU also actively monitors all four power rail voltages and eight temperature sensors located all over both sides of the module. This chapter details the overall maintenance architecture.

## 7.1 Debugging and System Monitoring

The host computers is responsible for assigning IP addresses to each module, and hence for maintaining a mapping of module location within the machine to IP address. This map allows images generated by the program mapper to be loaded into the correct modules at boot time, and it allows for seamless replacement of malfunctioning modules.

The host computer communicates with each module over TCP/IP channels. We use a simple ASCII-based protocol for establishing connectivity, handling error conditions, and monitoring power and temperature conditions. In order to support programming and in-system debugging, we use a simple byte-oriented protocol to stream the industry standard XSVF files over the channel. The MCU then interprets the XSVF data and issues the corresponding JTAG TAP commands to the target device [59]. Using XSVF-based communication facilitates in-system debugging via both Analog Devices, and Xilinx development suites.

On a module, the Wiznet W3150A TCP/IP stack is configured to maintain four TCP/IP channels with separate buffers. One channel is dedicated to critical error handling. The second channel is used for estab-

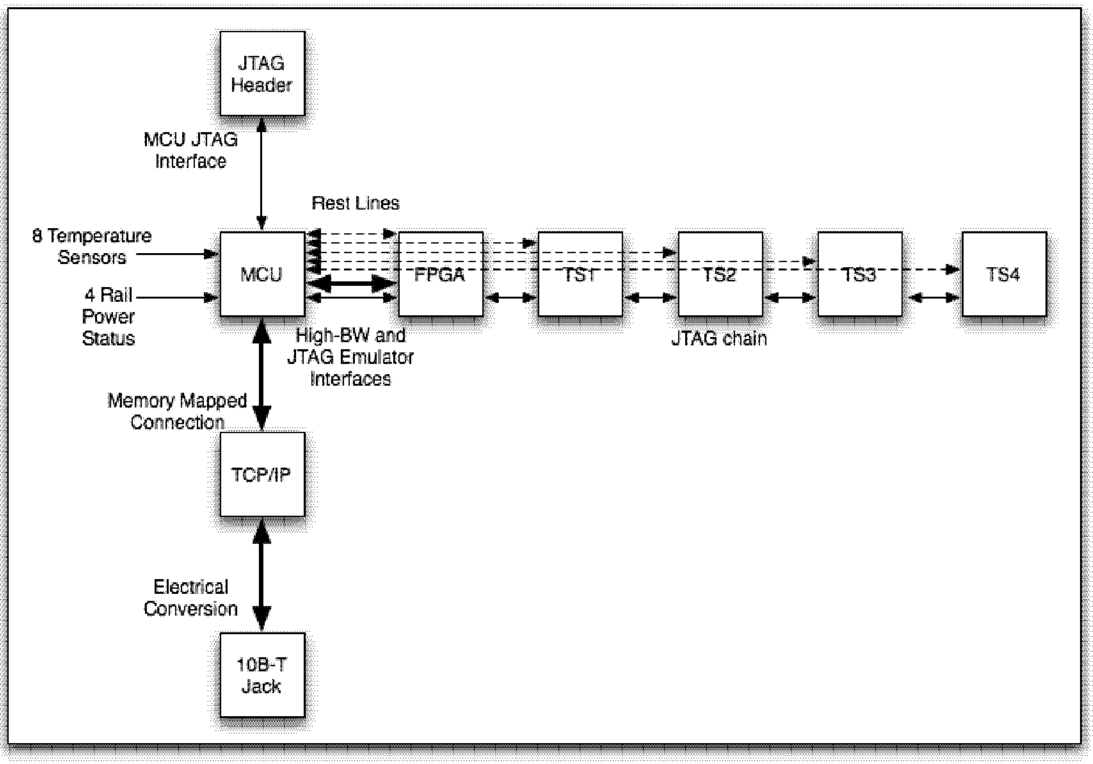

**Figure 7.1: Maintenance Interconnections**

lishing connectivity and monitoring the system. The third channel is used for XSVF based communication, and the last channel is not currently utilized. Using separate channels with separate buffers greatly increases the robustness of the interface in the event of deadlock due to critical errors.

## 7.2 Booting

Upon first applying power to a module, the MCU begins executing module initialization code that:

- holds the FPGA in reset
- holds the DSPs in reset
- monitors the four power rails
- monitors eight temperature sensors placed throughout the module
- initializes the hardware TCP/IP stack

- acquires an IP address from the host
- awaits host commands

The host eventually streams the FPGA bit-programming file to the MCU, which the MCU interprets and converts to JTAG TAP commands in order to program the FPGA. The MCU then reads the bit-stream back out of the programmed FPGA and streams it back to the host for host-side verification. After the host verifies that the FPGA has been programmed correctly, it instructs the MCU to bring the TigerSHARCs out of the reset state. The host then sends the TigerSHARC boot kernel to the MCU, which the MCU streams into the FPGA; the FPGA then boots the TigerSHARCs through the link-ports [4] via the TigerSHARC Physical Layer. After the initial booting of the TigerSHARCs completes, a similar sequence is followed to load the full program text and data segments into the DSPs and RAM. After the initial programming phase completes, the FPGA initializes the MGT and I/O channels and informs the MCU of the communication channel status. When all communications channels are functioning, the host allows the modules to begin computation.

# Chapter 8

# Programming Model and Applications

This thesis details the entire physical design and architecture of the ILLIAC 6, but without a programming model and an example application, it is difficult to understand how powerful a single module can be and how elegantly the platform scales with greater application demands. In this chapter, we provide an overview of how ILLIAC 6 programs are generated and mapped to the physical machine, we detail a real example application, and we describe the contracts between the programming model and the hardware and explain how we guarantee deterministic bandwidths.

## 8.1 Programming Model

We have chosen a *component-based* programming model for the ILLIAC 6 for several reasons. First, it represents a clear division of responsibility between parallelization of a component and management of computational and communication resources. Second, it facilitates a seamless programming paradigm that includes both processors and FPGA fabric which builds on the rich array of tools already available for programming these devices. Third, it allows the programmer to completely abstract the topology of the machine away from the application; the machine topology is the exclusive concern of the program mapper.

In our terminology, a component is a computational unit with typed input and ouptut streams. For example, a component that compresses voice data and packages it as IP packets might have an input stream of type *sample block* and an output stream of type *IP packet*. Furthermore, components may maintain state; this consideration is internal to the component and is not the concern of the program mapper.

Components are registered in libraries in order to make them available for use in applications. Figure 8.1 shows a simple library with three components. Component $A$ takes input streams of type $R$ and $S$ and produces a stream of type $T$; Components $B$ and $C$ perform related stream operations. Semantically, these components represent abstract capabilities; they can utilize either DSP resources, FPGA resources, or both, to perform computation. The program mapper can configure them to use different types of resources depending on demanded bandwidth.

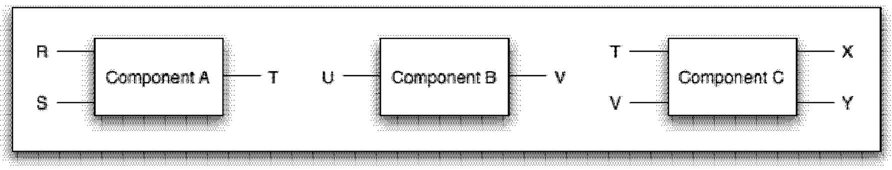

**Figure 8.1: A Component Library**

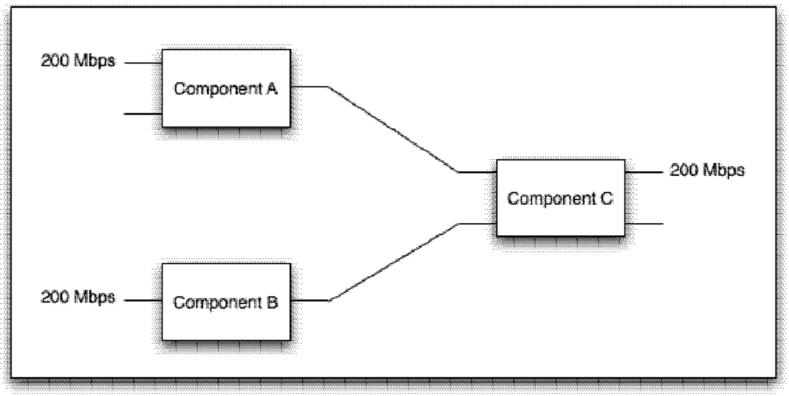

**Figure 8.2: An Unrealized Application**

Registered components can be wired into an application. Figure 8.2 shows an application in which the three components have been connected to form a program that takes input streams of type $R$, $S$, and $U$, and produces output streams of type $X$ and $Y$. Not shown here is a specification of the input and output connectivity of the application – where it gets its input data and where it sends its outputs. In addition to the topology of the application, the developer specifies bandwidth requirements for the input and output streams. In this case, $R$s and $U$s arrive at the rate of 200 Mbps, and $X$s must be produced at the rate of 200 Mbps. We refer to the application as unrealized at this point, because the bandwidth requirements may be infeasible for the requested components given the currently available machine resources.

To be registered in a library, a component must provide an interface to the program mapper that describes the component's type and that permits the program mapper to convey a performance requirement to the component. The performance requirement is a set of minimum bandwidth requirements imposed on the inputs and/or outputs of the component. The program mapper configure the component to realize the requested bandwidth by replacing the component with a subgraph of concrete components that demand certain computational resources: processor bandwidth, FPGA fabric, and communication bandwidth internal to the component. Figure 8.3 shows the example application in realized form. Component $A$ has been

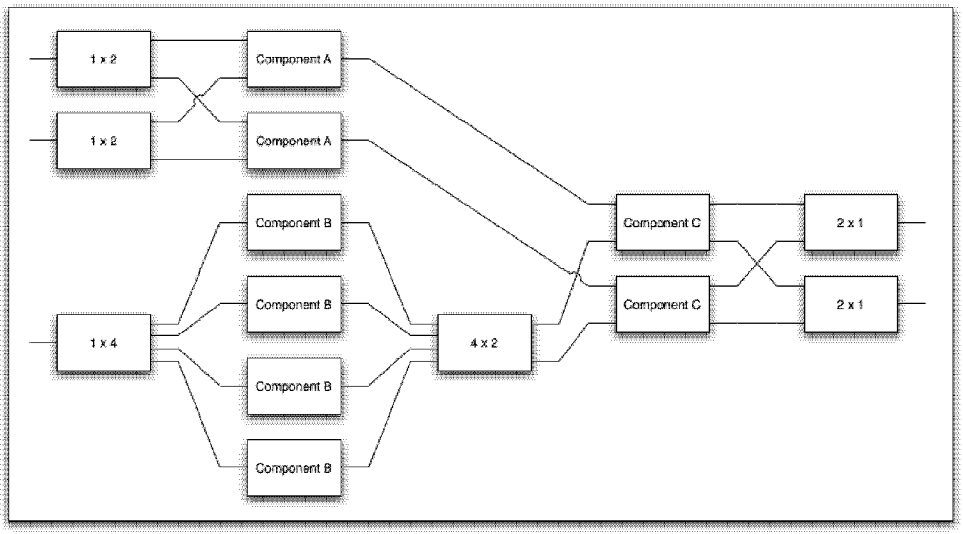

**Figure 8.3: A Realized Application**

replaced by two concrete components that can accept 100 Mbps each, and the other components have been similarly replaced. Connectors – data stream demultiplexers and multiplexers – are added by the components so that their external interfaces and original semantics are maintained. These data stream connectors are known to the program mapper and can be combined and simplified.

The program mapper is responsible for mapping the realized application onto the target system. The interconnection topology of the system is given as an input to the program mapper, along with the topology of the realized application and its resource requirements. The mapper is responsible for the allocation of computational bandwidth, FPGA fabric, interconnect bandwidth, and power. The direct allocation of power as a resource in the program mapper is made possible by the coordinated design of the mezzanine card and program mapper. The FPGA clock rate and the TigerSHARC clock rates can be reduced down to the minimum rate required to achieve the bandwidth demands of the program mapper.

## 8.2 Software Defined Radio NTSC TV Tuner

An SDR "is a radio communication system which uses software for the modulation and demodulation of radio signals" [57]. Our first *real* application is a novel SDR that maps very naturally to a single ILLIAC 6 module. The application simultaneously tunes to four different NTSC analog television stations, demodulates each channel, performs scan conversion to RGB based frames, combines the four channels into a single four-quadrant channel, and outputs the stream as 480p digital video via a DVI connector. Figure 8.4 details the

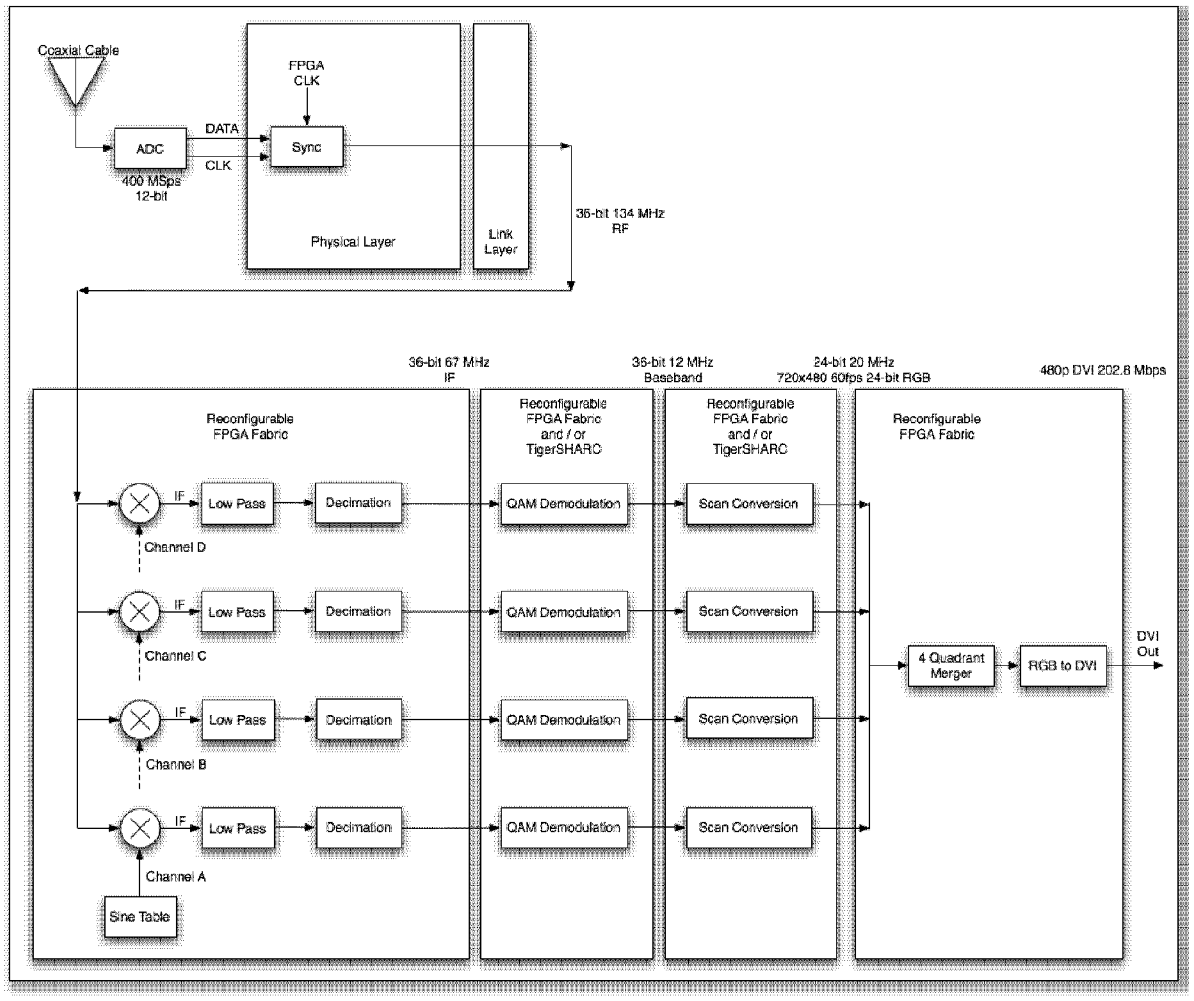

**Figure 8.4: Application Architecture of a 4-Channel All Digital NTSC TV Tuner and Demodulator to DVI**

application architecture.

One of the novel aspects of this application is the lack of an analog front-end that performs RF to IF conversion. We simply use a high-quality ADC and perform the IF conversion digitally within the FPGA fabric. Furthermore, we utilize a standard CATV feed as the signal input removing the need for analog amplification and tuning traditionally found in an analog front-end. The only analog elements required in this application are the CATV feed, a passive dongle to match the impedance of the coaxial cable to the ADC input, an analog low-pass filter built into the ADC to reduce aliasing, and the ADC itself.

Realizing this application within an ILLIAC 6 module is rather straightforward. The ADC streams data into a module via the 50 Gbps I/O header, and the ADC physical layer utilizes our asynchronous buffer primitive, detailed in Section 6.6.2, to reliably move the data into the FPGA fabric clock domain. The ADC link layer is vacuous, and the ingress computation layer performs the simple RF to IF conversion for each channel, essentially performing a stream fork operation; the single incoming stream is converted into four outgoing streams within this layer. The ingress computation layer is also responsible for framing and for adding the routing headers to the data stream. The choice of where and how to route the IF stream is actually an interesting question. The inherent flexibility provided by a module allows us to perform the QAM demodulation and the NTSC to RGB scan conversion in either the DSPs, the FPGA, or both. The module may be configured to run several other applications, so this flexibility in component location is essential to maximizing the potential of a module.

QAM demodulation is routinely performed in DSPs, and a number of hardware QAM demodulators have been demonstrated and can be easily ported to work within the FPGA fabric [24] [58]. Hence, creating logically equivalent QAM demodulation components realized in FPGA fabric or as code running on a DSP is not difficult. The bandwidths of these components may differ, but they can be arbitrarily dilated as detailed in Section 8 in order to satisfy the application bandwidth constraints. Similar argument hold for the scan conversion component.

After the streams have been scan-converted in the FPGA or the DSPs or both, they are merged together in the egress computation layer. Each video stream frame is reduced in size by a factor of four and placed in one of the four quadrants of the outgoing stream as shown in Figure 8.5. The egress computation layer also converts the RGB-based streams to the industry standard DVI format at a resolution of 480p.

## 8.3 Ensuring Deterministic Bandwidth

Our programming model requires that every component have bandwidth guarantees on its ingress and egress ports, and it requires that all communication channels guarantee the bandwidth request by the mapper. Providing bandwidth guarantees for a reconfigurable component realized in the FPGA fabric is straightforward, because architecturally every component will have an upper bound on the number of clock cycles it requires to perform a computation. The program mapper controls the component clock rate, and hence will always be able to ensure deterministic bandwidths for an FPGA-based component. Ensuring deterministic bandwidths for a component realized within a TigerSHARC is more complicated, because computations in a DSP can be highly data-dependent, and the component may rely on external memory.

Even when a component utilizes highly data-dependent computations, it is always possible to determine the upper bound on the number of clock cycles required to perform a computation. It is also relatively

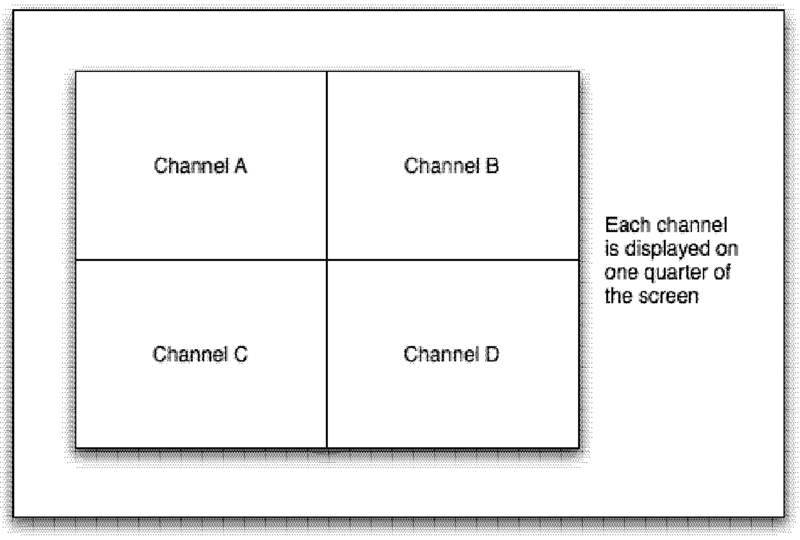

**Figure 8.5: 4-Channel All Digital NTSC TV Tuner and Demodulator Output**

straightforward to generate and use profiling data to find the mean number of clock cycles required along with an expected variance. Using the upper bound will always result in a hard computation bandwidth guarantee, but it may grossly under utilize the processor. Using the mean and variance along with the buffer depths on the ingress and egress channels attached to the component, it is possible to generate a bandwidth distribution that can easily be used to provide soft computation bandwidth guarantees. Whether hard or soft guarantees are appropriate is both application and I/O dependent. For example, if the I/O attachment sourcing or sinking data supports flow control, or if dropping data is acceptable, such as with many video streams, then soft guarantees are an appropriate choice.

An important aspect of our hardware design throughout every level of the system hierarchy has been deterministic access latencies and bandwidths. Each DSP has its own dedicated external memory interface and banks, so when components utilize external memory, it is a much simpler matter to provide bandwidth guarantees than if the DSPs shared memory banks. Moreover, an important aspect of our DSP to FPGA interface is that it has a deterministic access latency, so much like with external memory, providing bandwidth guarantees for link-port communication is straightforward.

Providing bandwidth guarantees for entire communication channels is a difficult problem. The link-port interface from a DSP to the FPGA is deterministic, but we must also ensure the stream bandwidth within the FPGA communication stack. This is a difficult task, because the underlying channels are lossy and require retransmission based protocol as described in Section 6.3, and the crossbar switch in the network layer of the FPGA must handle stream arbitration and routing deterministically.

Over the longest channel, a 30 ft link, at 10 Gbps, and at a realistic signal velocity of 85 ps/in, at most 2 frames can reside on the wire at any given time. If the MGT link layer has a latency of 1 frame, then up to 8 frames may need to be retransmitted in the event of bit errors in a single frame. The expected channel BER is approximately $10^{-10}$, but under the worst possible conditions, we can say that pessimistically the BER might fall to $10^{-7}$ for some short duration of time. If we assume a uniform distribution of errors, then the retransmission overhead at a BER of $10^{-7}$ will result in an 8 frame retransmissions every 625 frames, which is an overhead of 1.28%. It is clear that even under the worst possible conditions, that the bandwidth loss from ensuring reliably communication is negligible, and can be entirely accounted for in a hard bandwidth guarantee model by underutilizing each channel by 1.28%.

The bandwidth guarantees provided by the crossbar arbitration policy are more difficult to analyze. It is clear that the EDF policy detailed in Section 6.5 will provide hard bandwidth guarantees for all streams if the streams do not exceed their requested bandwidths. This can be easily achieved for streams originating within the system but may be more difficult for I/O originated streams. If an upper bound on the bandwidth of I/O streams is used then every stream can be assured its required bandwidth at the network switch. However if a bandwidth average is used, then it is difficult to provide a bandwidth guarantee for any other stream in the system unless it has a disjoint path. The critical parameters in this situation are the message sizes. If a stream uses enormous messages that are much larger than the network buffers and saturates the channel with a higher data rate than requested, other streams may not be given their requested bandwidths. However, given the message sizes, and bandwidth distributions, it is possible to provide soft bandwidth guarantees for all streams. On the other hand, if all messages fit within the buffers at each node, then we expect the EDF policy to still provide bandwidth guarantees for all streams, but we have not proved this assumption.

# Chapter 9

# Related Work

Numerous high-performance computing systems have employed shuffle-exchange topologies, including the NYU Ultracomputer [28], BBN Butterfly [33], and Cedar [10], [50]. Multiprocessor signal processing systems have been developed commercially; Mercury Computer Systems has developed several multiprocessor signal processing systems for example.

Component-based programming has been explored in both industrial and research settings, particularly in the case of signal processing systems. The SCE framework [41] is similar to ours in that components are instantiated in light of performance demands. But SCE components are far more general and transportable than ours and have a more complex, object-oriented flavor.

Programming models for real-time, high-throughput applications have been developed. In [34], the Intel IXP is managed by a compilation system that was the inspiration for the ILLIAC 6 program mapper. IXP C components (called Packed Processing Stages) are sequential components which communicate via pipes (ring buffers). They are mapped onto the physical resources of the IXP (cores, memory bandwidth, etc.) by the compilation system, which uses static simulation to evaluate mappings. The user provides a performance target to the compilation system. This approach cannot be applied to the large applications we envision for ILLIAC 6 because automatic parallelization of the type used in [34] breaks down on large and complex programs. For this reason we prefer a model in which the components themselves assume more responsibility for achieving the targeted bandwidth.

Streaming languages [14], [27], [37], [48] have been developed for a variety of hardware types and application domains, including signal processors, graphics processors, and network processors. These languages are closely related to our component model.

Systems for automatic generation of DFTs [43] and linear algebra libraries [9] can be used to automatically instantiate components for our systems. At the same time the very sophistication of these generators speaks to the difficulty of fully automating the task of component instantiation (i.e., automatic parallelization and optimization).

There are a number of commercial and academic reconfigurable computing systems being designed and developed. Some of these systems leverage FPGAs as a reconfigurable coprocessor for commercial-off-the-shelf processors, and some of the systems utilize only FPGAs. Many of the mixed FPGA and processor systems are bandwidth-limited between the CPU and the FPGA. However, some of these systems such as the Cray XD1 [15] provide an extremely high-bandwidth CPU to FPGA bus. Unfortunately these systems cannot provide a deterministic access latency between these devices, so their application to real-time systems is extremely limited. Some of the FPGA-only systems such as the BEE2 [12] address this non-determinisitc access latency issue, but they do so at increased board price. The BEE2 module contains five large FPGAs, four of which are available for reconfigurable computation, but it cost approximately $20,000 in 2005 [12]. The ILLIAC 6 module provides a high-bandwidth FPGA to DSP connection at the low cost of approximately $1000 per module. In the commercial arena, both Nallatech and SRC Computers have designed and developed a number of reconfigurable computing systems utilizing Virtex-2 Pro and Virtex-4 FPGAs. Unfortunately, since they are commercial systems, few details on their physical architectures or performance are available.

It is difficult to compare the performance of modern high-performance computing platforms, because each platform offers a customized combination of memory, reconfigurable silicon, FPGA embedded processors, I/O, and COTS CPUs or DSPs. Moreover, each platform provides wildly varying programming models, and application mappers, so only the simplest of computational kernels can be used to provide some relative performance metrics. Unfortunately, it is widely agreed upon in the architecture community that the most useful and meaningful metrics for comparing platform performances are obtained by using kernels from a wide selection of real applications. As we cannot obtain this data, we provide a comparison of the key computational resources that are available in two comparable systems. The IBM Blue Gene/L node card [26] contains 16 nodes with two custom PowerPC processors on each node, and the BEE2 module [12] contains five large Virtex-2 Pro FPGAs. We compare these two systems to an ILLIAC 6 carrier board, which contain eight modules with four TigerSHARC DSPs and one Virtex-2 Pro FPGA on each module in Table 9.1.

It is clear from Table 9.1 that these three systems are similar in size and power consumption, but both the ILLIAC 6 and BEE2 cards have a substantial I/O bandwidth advantage as compared to the Blue Gene/L blade. In terms of reconfigurable computing power, the ILLIAC 6 board has similar FPGA resources to the BEE2, but the Blue Gene/L has no reconfigurable computing elements. The processor computing power of these systems is also difficult to compare, because the Blue Gene/L is designed to excel at floating point operations, whereas the ILLIAC 6 and the BEE2 excel at integer operations. The Blue Gene/L node card can perform up to 180 billion FLOPS, and the TigerSHARCs on the ILLIAC 6 carrier board can perfom up to 153.6 billion MACs per second. The 10 embedded PowerPC 405 cores in the BEE2 module can

| **Computer** | **Reconfigurable Silicon** | **Dedicated Processors** | **RAM** | **I/O Bandwidth** | **Power Consumption** |
|---|---|---|---|---|---|
| ILLIAC 6 Carrier Board | 246,528 Logic Cells & 1,088 18 x 18 Multiplier Blocks | 16 Embedded PowerPC 405 Cores & 32 TigerSHARC DSPs | 4 GB | 360 Gbps | ≈ 250 W |
| BEE2 Module | 372,240 Logic Cells & 1,640 18 x 18 Multiplier Blocks | 10 Embedded PowerPC 405 Cores | 20 GB | 360 Gbps | ≈ 250 W |
| Blue Gene/L Node Card | N/A | 32 Custom PowerPC 440 ASICs | 16 GB | 67.2 Gbps | ≈ 250 W |

**Table 9.1: Resource Comparison Between an ILLIAC 6 Carrier Board, a BEE2 Module, and a Blue Gene/L Node Card**

perform up to 6,000 MIPS, and although we have not provided a programming framework to facilitate using the embedded PowerPC 405 cores in the ILLIAC 6, they can be utilized by the ILLIAC 6 reconfigurable computation layer, so they are included in our resource comparison. The 16 embedded PowerPC 405 cores in the ILLIAC 6 carrier board can perform up to 9,600 MIPS. Finally, both the BEE2 and the Blue Gene/L systems have similar memory resources, but the ILLIAC 6 has appreciably less RAM.

Finally, it is interesting to compare a full ILLIAC 6 system to a full Blue Gene/L system, as a full ILLIAC 6 contains 65,536 DSPs, and a full Blue Gene/L contains 65,536 nodes. The ILLIAC 6 is not designed to compete with floating point computational power of the Blue Gene/L, which can perform 360 trillion FLOPS when using both node processors as computing elements and 180 trillion FLOPS when configured for applications requiring one of the node processors to act as the communications controllers. However, the ILLIAC 6 can actually outperform the Blue Gene/L in its typical configuration. Counting only the computational power of the DSPs, the ILLIAC 6 can perform 236 trillion FLOPS. More importantly, a full ILLIAC 6 can perform 314 trillion 16-bit MACs per second solely using the DSPs. Including the reconfigurable FPGA fabric in these computations will dramatically increase the performance. Furthermore, the ILLIAC 6 aggregate I/O bandwidth of 11.5 TBps is approximately two order of magnitude greater than the Blue Gene/L aggregate I/O bandwidth of 128 GBps. Lastly, a full ILLIAC 6 consumes approximately 500 KW, while the full Blue Gene/L consumes approximately twice as much power, 1 MW.

# Chapter 10

# Conclusions and Future Work

In this thesis we provide an overview of the ILLIAC 6 platform and its physical design. We architected the system from the ground up to provide enormous computational and communication resources, to seamlessly scale from a single module to a full system, and to provide robust, error free communication channels at every level in the hierarchy. We provided a good mixture of both traditional DSP resources and reconfigurable silicon to the application developer, and we have developed a simple but powerful programming model.

We describe in detail the physical design of a module and a carrier board, and we explain the important aspects of the larger system design. We provide numerous details of how we ensure signal integrity throughout the machine, and we include simulation results in the form of eye diagrams for all of the high-speed channels and significant interfaces.

We designed a novel communications stack which directly incorporates reconfigurable silicon into the communications architecture, and we have provided deterministic high-bandwidth I/O connections between the DSPs and the reconfigurable silicon. We have provided primitives to perform the difficult task of reliably synchronizing to I/O devices, and we detail the contracts between the program mapper and the hardware necessary to ensure bandwidth constraints. We also discuss a number of related works and provide a brief comparison between a carrier card and a two other modern high-performance computers. Finally, we detail a robust means of booting, programming, and managing any size machine, and we detailed an example application and its mapping to a module.

We are currently in the process of completing the module layout. We have finished schematic capture, we have ordered all of the board components, and we have chosen a manufacturer and assembler. We are in the final stages of routing the PCB and expect to have the first assembled modules in August 2006. After testing and debugging the modules, we will begin the layout of a carrier board followed by a midplane. Certain aspects of the chassis design, including power regulation, thermal management, and physical construction remain open problems for further exploration.

Once we have working modules, we will begin exploring a number of interesting aspects of the reconfigurable silicon computation layers, and we will begin designing FGPA and DSP component primitives and libraries. We will also begin benchmarking the design and profiling applications.

Therefore, as we have shown in this thesis, the ILLIAC 6 is a novel reconfigurable supercomputing platform which addresses a unique class of modern problems. We are confident that as our system design continues, we will be able to make substantial contributions to the field of reconfigurable computing.

# References

[1] *170 MHz FIFOs Using the Virtex Block SelectRAM+ Feature*, XAPP131 v1.7, Mar. 26, 2003.

[2] J. Adam. New correlations between electrical current and temperature rise in pcb traces. In *Semiconductor Thermal Measurement and Management Symposium, 2004*, pages 292–299. IEEE Computer Society Press, 2004.

[3] *ADSP-TS201S TigerSHARC®Embedded Processor Data Sheet*, Rev. 0, 2004.

[4] *ADSP-TS201S TigerSHARC®Processor Hardware Reference*, Rev. 1.1, 2004.

[5] *ADSP-TS201S TigerSHARC®Processor Programming Reference*, Rev. 1.0, 2004.

[6] *AdvancedTCA PICMG 3.0 Short Form Specification*, Jan. 2003.

[7] David L. Andrews, Douglas Niehaus, Razali Jidin, Michael Finley, Wesley Peck, Michael Frisbie, Jorge L. Ortiz, Ed Komp, and Peter J. Ashenden. Programming models for hybrid FPGA-CPU computational components: A missing link. *IEEE Micro*, 24(4):42–53, 2004.

[8] *ATmega640/V - ATmega1280/V - ATmega1281/V - ATmega2560/V - ATmega2561/V Preliminary Data Sheet*, Jun. 2006.

[9] Automatically tuned linear algebra software (atlas).

[10] John Beetem, Monty Denneau, and Don Weingarten. The gf11 supercomputer. *SIGARCH Comput. Archit. News*, 13(3):108–115, 1985.

[11] Kiran Bondalapati and Viktor K. Prasanna. Reconfigurable computing systems, November 12 2002.

[12] Chen Chang, John Wawrzynek, and Robert W. Brodersen. BEE2: A High-End Reconfigurable Computing System. *IEEE Design & Test of Computers*, 22(2):114–125, March/April 2005.

[13] Compton and Hauck. Reconfigurable computing: A survey of systems and software. *CSURV: Computing Surveys*, 34, 2002.

[14] Corinna Cortes, Kathleen Fisher, Daryl Pregibon, Anne Rogers, and Frederick Smith. Hancock: A language for analyzing transactional data streams. *ACM Trans. Program. Lang. Syst.*, 26(2):301–338, 2004.

[15] *Cray XD1 Datasheet Release 1.3*, 2005.

[16] *Cypress CY23EP05 - 2.5V or 3.3V, 10-220 MHz, Low-Jitter, 5-Output Zero Delay Buffer*, 38-07759 Rev. *B, Dec. 13, 2005.

[17] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[18] *Data-Width Conversion FIFOs Using the Virtex-II Block RAM Memory*, XAPP261 v1.0, Jan. 10, 2001.

[19] Martyn Edwards and Peter Green. An object oriented design method for reconfigurable computing systems. In *DATE*, pages 692–696. IEEE Computer Society, 2000.

[20] *Engineer-to-Engineer Note EE-170 - Estimating Power for ADSP-TS201S TigerSHARC®Processors*, Rev. 1, Jan. 3, 2005.

[21] *Engineer-to-Engineer Note EE-179 - ADSP-TS201S TigerSHARC®System Design Guidelines*, Rev. 5, Aug. 12, 2005.

[22] *Engineer-to-Engineer Note EE-261 - Understanding Jitter Requirements of PLL-Based Processors*, Rev. 1, Jan. 20, 2005.

[23] *Engineer-to-Engineer Note EE-68 - Analog Devices JTAG Emulation Technical Reference*, Rev. 9, Oct. 18, 2004.

[24] Farzad Etema155di. Design and implementation of a 155.52 mbps atm transceiver.

[25] *FIFOs Using Virtex-II Block RAM*, XAPP258 v1.4, Jan. 7, 2005.

[26] Gara, A. et al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2/3):195–212, 2005.

[27] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, Jeremy Wong, Henry Hoffmann, David Maze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 291–303, New York, NY, USA, 2002. ACM Press.

[28] Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir. The nyu ultracomputer: designing a mimd, shared-memory parallel machine (extended abstract). In *ISCA '82: Proceedings of the 9th annual symposium on Computer Architecture*, pages 27–42, Los Alamitos, CA, USA, 1982. IEEE Computer Society Press.

[29] H. Johnson and M. Graham. *High-Speed Digital Design: A Handbook of black Magic*. Cambridge University Press, 1998.

[30] H. Johnson and M. Graham. *High Speed Signal Propagation: Advanced Black Magic*. Prentice Hall, 2003.

[31] *Interfacing Virtex-II Series FPGAs With Analog Devices TigerSHARC TS20x DSPs via LVDS Link Ports*, XAPP635 v1.1, Feb. 23, 2005.

[32] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. pages 129–139.

[33] Thomas J. LeBlanc, Michael L. Scott, and Christopher M. Brown. Large-scale parallel programming: experience with bbn butterfly parallel processor. In *PPEALS '88: Proceedings of the ACM/SIGPLAN conference on Parallel programming: experience with applications, languages and systems*, pages 161–172, New York, NY, USA, 1988. ACM Press.

[34] Long Li, Bo Huang, Jinquan Dai, and Luddy Harrison. Automatic multithreading and multiprocessing of c programs for ixp. In *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 132–141, New York, NY, USA, 2005. ACM Press.

[35] *LogiCORE Aurora Data Sheet*, DS128 v2.4, Jan. 10, 2006.

[36] *Linear Technology: LTC1628-SYNC - High Efficiency, 2-Phase Synchronous Step-Down Switching Regulator*, LT 1205 Rev. A, 2005.

[37] William R. Mark, R. Steven Glanville, Kurt Akeley, and Mark J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph.*, 22(3):896–907, 2003.

[38] *Maxim 12.5Gbps Settable Receive Equalizer - MAX3804*, 19-2713 Rev. 1, Nov. 2003.

[39] *Maxim LVTTL/TTL-to-Differential LVPECL/PECL Translators - MAX9372*, 19-2377 Rev. 0, Apr. 2002.

[40] *Maxim Multiple-Output Clock Generator with Spread Spectrum - MAX9492*, 19-3683 Rev. 0, May 2005.

[41] DARPA BAA 00-59 Polymorphic Computing Architecures (PCA), Milestone Q2,Q3,Q4 Report. 2002.

[42] *Metastability Recovery in Virtex-II FPGAs*, XAPP094 v3.0, Feb. 10, 2005.

[43] Peter A. Milder, Mohammad Ahmad, James C. Hoe, and Markus Pueschel. Fast and accurate resource estimation of automatically generated custom dft ip cores. In *FPGA'06: Proceedings of the internation symposium on Field programmable gate arrays*, pages 211–220, New York, NY, USA, 2006. ACM Press.

[44] T. Mudge. Power: a first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[45] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[46] *RocketIO™ Transceiver User Guide*, UG024 v2.3.1, May 20, 2004.

[47] *Samsung - K4M51323LC - S(D)N/G/L/F Mobile-SDRAM*, Mar. 2006.

[48] Niraj Shah, William Plishker, and Kurt Keutzer. Np-click: A programming model for the intel ixp1200. In *2nd Workshop on Network Processors (NP-2) at the 9th International Symposium on High Performance Computer Architecture (HPCA-9)*, 2003.

[49] *Synchronous and Asynchronous FIFO Designs*, XAPP051 v2.0, Sep. 17, 1996.

[50] Josep Torrellas and Zheng Zhang. The performance of the cedar multistage switching network. In *Supercomputing '94: Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 265–274, New York, NY, USA, 1994. ACM Press.

[51] *Virtex-4 Family Overview*, DS112 v1.5, Feb. 10, 2006.

[52] *Virtex-II Pro / Virtex-II Pro X - 3.3V I/O Design Guidelines*, XAPP659 v1.6, Jun. 9, 2005.

[53] *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, UG012 v4.0, Mar. 23, 2005.

[54] *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, DS083 v4.5, Oct. 10, 2005.

[55] W. Dally and J. Poulton. *Digital Systems Engineering*. Prentice Hall, 1993.

[56] *W3150A Data Sheet*, v1.0.2, Dec. 28, 2005.

[57] Wikipedia. Software radio — wikipedia, the free encyclopedia, 2004. [Online; accessed 3-July-2006].

[58] B. C. Wong and H. Samueli. A 200-mhz all-digital qam modulator and demodulator in 1.2-$\mu$m cmos for digital radio applications. *IEEE Journal of Solid-State Circuits*, 26(12):1970–1980, 1991.

[59] *Xilinx In-System Programming Using an Embedded Microcontroller*, XAPP058 v3.1, Jun. 25, 2004.